



Escola d'Enginyeria de Telecomunicació i  
Aeroespacial de Castelldefels

UNIVERSITAT POLITÈCNICA DE CATALUNYA

# TRABAJO FINAL DE GRADO

**TÍTULO DEL TFG:** Anonimato en blockchain: Monero

**TITULACIÓN:** Grado en Ingeniería Telemática

**AUTOR:** Samuel Segura Ramírez

**DIRECTOR:** Olga León Abarca

**FECHA:** 7 de julio de 2020



**Título:** Anonimato en blockchain: Monero

**Autor:** Samuel Segura Ramírez

**Director:** Olga León Abarca

**Fecha:** 7 de julio de 2020

## **Resumen**

Blockchain es una base de datos descentralizada que registra bloques de información y los entrelaza mediante funciones criptográficas, de manera que se garantiza la integridad de dicha información. La principal aplicación de blockchain son las transacciones monetarias, y de hecho cada blockchain utiliza una divisa propia para la mismas. La primera y más conocida de estas divisas, mejor conocidas como criptomonedas, fue Bitcoin.

Una de las ventajas de usar Bitcoin es que no requiere el uso de una entidad central que verifique los datos de las transacciones y los apruebe. En las transacciones tradicionales los bancos son los encargados de ejercer este rol. En la blockchain, los propios usuarios verifican los datos de dichas transacciones y usan un mecanismo de consenso que permite aprobar o no cada transacción. A pesar de ello, esta criptomoneda presenta diferentes problemas, como la falta de anonimato, y la comunidad científica empezó a desarrollar otras tecnologías blockchain con el objetivo de mejorar los puntos débiles de Bitcoin.

Monero es una blockchain creada en 2014 que utiliza criptografía avanzada para proporcionar un elevado nivel de anonimato a sus usuarios. El objetivo de Monero es evitar que terceras partes puedan relacionar una transacción con el remitente y el destinatario de la misma. Para ello hace uso de mecanismos como la firma en anillo, la división del gasto o el uso de dos pares de clave pública privada diferentes.

Un estudio realizado en 2017 analizó la posibilidad de desactivar el anonimato que proporciona la firma en anillo. Como consecuencia, se incluyó una nueva característica en la blockchain denominada RingCT, que permite ocultar la cantidad de dinero enviada en las transacciones.

Los objetivos de este proyecto son por un lado analizar el funcionamiento de Monero de forma teórica y práctica, poniendo en marcha una blockchain de test. En segundo lugar, se pretende analizar usando algoritmos de Machine Learning, bajo qué circunstancias un atacante puede sobrepasar los mecanismos de seguridad como RingCT y romper el anonimato en la blockchain.

**Title:** Anonymity in blockchain: Monero

**Author:** Samuel Segura Ramírez

**Director:** Olga León Abarca

**Date:** July 7th 2020

## **Overview**

Blockchain is a decentralized database that records blocks of information and links one block to the previous ones by means of cryptographic functions. In this way, information integrity is guaranteed. So far, the main application of blockchains have been monetary transactions, and usually each blockchain uses its own currency. The first and best-known of these currencies, better known as cryptocurrencies, was Bitcoin.

One of the advantages of using Bitcoin against traditional monetary systems is that it does not require the presence of a central entity, such a bank, in charge of verifying data and validating transactions. In a blockchain, its own users are responsible for that and make use of a consensus protocol to approve or reject transactions. Despite it, this cryptocurrency exhibits many problems such as the lack of anonymity in the transactions. For this reason, the scientific community has developed other approaches in order to overcome Bitcoin limitations.

Monero is a blockchain created in 2014, which uses advanced cryptography in order to provide a high degree of anonymity to its users. Its main goal is to prevent third parties from linking a transaction with its sender and its receiver. With this purpose, it makes use of several security mechanisms such as ring signature, the division of the expense and two pairs of public-private keys or RingCT, a feature that allows you to hide the amount of money sent in transactions.

A recent study analyzed the likelihood of cancelling anonymity provided by the ring signature. As a consequence, a new feature named RingCT was added to Monero, which allows the hiding of the amount of cryptocurrency sent in a specific transaction.

The goals of this work are twofold. On the one hand, to analyze the performance of Monero in a theoretical and practical way, by running a test blockchain. On the other hand, we aim at analyzing by means of Machine Learning algorithms under which circumstances an attacker can overcome security mechanisms such as RingCT and break Monero's anonymity.

# ÍNDICE

<b>LISTA DE FIGURAS</b>	<b>7</b>
<b>ACRÓNIMOS</b>	<b>9</b>
<b>INTRODUCCIÓN</b>	<b>10</b>
<b>CAPÍTULO 1. BLOCKCHAIN Y MONERO</b>	<b>12</b>
1.1. La cadena de bloques	13
1.2. Bitcoin	15
1.2.1. Bloque de Bitcoin	15
1.2.2. Problemas de Bitcoin	19
1.2.2.1. Problemas técnicos	19
1.2.2.2. Problemas sociales	19
1.3. Monero	20
1.3.1. Características de Monero	21
1.3.1.1. Cuatro claves en lugar de dos	21
1.3.1.2. Firma en anillo	22
1.3.1.3. Subdirecciones	23
1.3.1.4. RingCT	24
1.3.2. Vulnerabilidades de Monero	25
1.3.2.1. Ataque del 51%	25
1.3.2.2. Cryptojacking	26
1.3.2.3. Trazabilidad antes y después de RingCT	29
<b>CAPÍTULO 2. ENTORNO EN MONERO TESTNET</b>	<b>30</b>
2.1. Monero Daemon	31
2.1.1. Monero Daemon RPC	33
2.2. Monero Wallet Command Line Interpreter	37
2.3. Monero Wallet RPC	39
2.4. Adaptación de hard forks en testnet	41
2.5. Desarrollo de un cliente RPC	42
2.5.1. Consumición de las APIs	44
2.5.2. Capa de persistencia	44
2.5.3. Capa de servicio	45
<b>CAPÍTULO 3. INTEGRACIÓN DEL CLIENTE RPC EN LA STAGE BLOCKCHAIN Y OBTENCIÓN DE DATOS</b>	<b>47</b>
3.1. Escenario	47
3.2. Reducción del anonimato usando outputs propios	48
3.2.1. Inyectar outputs	48
3.2.2. Guardar outputs	49

3.2.3. Guardar anillos existentes	50
3.2.4. Marcar outputs en anillos propios	51
3.2.5. Marcar outputs propios como señuelos	51
3.2.6. Clasificar anillos	52
3.2.7. Marcar outputs deducidos en otros anillos	53
3.3. Resultados de la inyección de outputs	54
3.4. Cálculo probabilístico	57
<b>CAPÍTULO 4. ANÁLISIS DE TRAZABILIDAD DE OUTPUTS</b>	<b>58</b>
4.1. Generar datasets	59
4.2. K-Nearest Neighbors	60
4.3. Decision Tree y Random Forest	61
4.4. Análisis teórico en mainnet	65
<b>CAPÍTULO 5. CONCLUSIONES Y DESARROLLO FUTURO</b>	<b>69</b>
<b>BIBLIOGRAFÍA</b>	<b>71</b>
<b>ANEXO A - TRANSACCIÓN EN MONERO</b>	<b>74</b>
<b>ANEXO B - OPCIONES DEL DEMONIO</b>	<b>76</b>
<b>ANEXO C - SCRIPTS PARA TRABAJAR CON WALLETS</b>	<b>78</b>

## LISTA DE FIGURAS

<b>Fig. 1.1</b>	Hashes en bloques consecutivos	14
<b>Fig. 1.2</b>	Cabecera de un bloque de Bitcoin	16
<b>Fig. 1.3</b>	Transacción de Bitcoin	17
<b>Fig. 1.4</b>	Inputs y outputs en una transacción de Bitcoin	18
<b>Fig. 1.5</b>	Cabecera de un bloque de Monero	21
<b>Fig. 1.6</b>	Anillo en transacción de Monero	23
<b>Fig. 1.7</b>	Transacción coinbase de Monero	24
<b>Fig. 1.8</b>	Transacción no coinbase de Monero	25
<b>Fig. 1.9</b>	Fork producido por un ataque del 51%	26
<b>Fig. 1.10</b>	Geografía del cryptojacking	28
<b>Fig. 2.1</b>	Escenario con tres nodos conectados	32
<b>Fig. 2.2</b>	Nodo 3 inicia y sincroniza con sus peers	33
<b>Fig. 2.3</b>	Salida del endpoint get_info	35
<b>Fig. 2.4</b>	Obtención de outputs con el endpoint get_outs	36
<b>Fig. 2.5</b>	Escenario con wallet conectada a nodo 1	37
<b>Fig. 2.6</b>	Wallet de Alice conectada al daemon local	38
<b>Fig. 2.7</b>	Wallet RPC conectada al daemon local	39
<b>Fig. 2.8</b>	Modificación para utilizar RingCT a partir de la altura deseada	41
<b>Fig. 2.9</b>	Obtención del primer output con amount ofuscado	42
<b>Fig. 2.10</b>	Escenario completo con APIs RPC y cliente	43
<b>Fig. 2.11</b>	Estructura de la aplicación desarrollada	43
<b>Fig. 2.12</b>	Diagrama UML de la base de datos	44
<b>Fig. 3.1</b>	Escenario completo en stagenet	48
<b>Fig. 3.2</b>	Método inject(n)	49
<b>Fig. 3.3</b>	Envío masivo de transacciones de 1 piconero	49
<b>Fig. 3.4</b>	Método persist_outputs(start_block)	50
<b>Fig. 3.5</b>	Método persist_rings()	50
<b>Fig. 3.6</b>	Método mark_my_rings()	51
<b>Fig. 3.7</b>	Método mark_my_outputs_in_other_rings(start_block)	52
<b>Fig. 3.8</b>	Método find_output_deducibility(start_block)	53
<b>Fig. 3.9</b>	Distribución del output real en anillos según el orden de edad	55
<b>Fig. 3.10</b>	Outputs restantes para la deducibilidad	56
<b>Fig. 3.11</b>	Cálculo probabilístico de deducibilidad	57
<b>Fig. 4.1</b>	Dataset de entrenamiento	60
<b>Fig. 4.2</b>	Clasificación con KNN	61
<b>Fig. 4.3</b>	Clasificación con Decision Tree	62

<b>Fig. 4.4</b> Tuning de parámetros de RF	63
<b>Fig. 4.5</b> Clasificación con RF	63
<b>Fig. 4.6</b> Anillos predichos con RF	64
<b>Fig. 4.7</b> Consulta de transacciones coinbase en wallet CLI	66
<b>Fig. 4.8</b> Cálculo de total de Monero invertido en stagenet	66
<b>Fig. 4.9</b> Valor de un monero en euros	67
<b>Fig. 5.1</b> Historial por años del tamaño de los anillos en Monero	70
<b>Fig. C.1</b> Script para crear una wallet	78
<b>Fig. C.2</b> Script para abrir una wallet	79



# ACRÓNIMOS

**API:** Application Programming Interface  
**ASIC:** Application-Specific Integrated Circuit  
**CLI:** Command Line Interface  
**CPU:** Central Processing Unit  
**DAO:** Data Access Object  
**DT:** Decision Tree  
**GPU:** Graphics Processing Unit  
**HTTP:** Hypertext Transfer Protocol  
**JSON:** Javascript Object Notation  
**KI:** Key Image  
**KNN:** K-Nearest Neighbors  
**ML:** Machine Learning  
**P2P:** Peer to Peer  
**PoW:** Proof of Work  
**RAM:** Random Access Memory  
**RF:** Random Forest  
**RingCT:** Ring Confidential Transactions  
**RPC:** Remote Procedure Call  
**SQL:** Structured Query Language  
**UML:** Unified Modeling Language  
**XMR:** Monero

## INTRODUCCIÓN

Durante los últimos años se ha popularizado un sistema de base de datos descentralizado llamado blockchain. Aparece en 2009 como una alternativa transparente e infalsificable a la banca tradicional, aunque sus usos no se limitan a las transacciones monetarias. Como su propio nombre indica, es una cadena de bloques. Los datos, en forma de transacciones, están contenidos en estos bloques, y la integridad e inmutabilidad de estos bloques están protegidas con funciones criptográficas. Los usuarios pueden mantener copias de la blockchain y contribuir al funcionamiento de ésta verificando transacciones y bloques a cambio de una recompensa.

Diversos tipos de aplicaciones han sido desarrolladas haciendo uso de blockchain, de las cuales las más extendidas han sido las criptodivisas. Estas criptodivisas aportan una serie de ventajas sobre el dinero tradicional, pero no están exentas de algunos problemas. La criptodivisa que fue creada originalmente junto a la blockchain fue Bitcoin. Esta fue la implementación más simple de una blockchain. Debido al potencial que ésta tenía, otras criptodivisas fueron creadas durante los años siguientes. Algunos ejemplos son Ethereum, la cual permite programar contratos inteligentes sobre su blockchain; o Litecoin, la cual mejora los tiempos de confirmación de transacciones. No obstante, estas blockchains presentan problemas de anonimato.

Monero es otra criptodivisa que fue creada entre otros motivos para solucionar este inconveniente. Utiliza varias capas de criptografía como firmas en anillo, dos pares de claves públicas o la posibilidad de utilizar subdirecciones. Concretamente, nos centraremos en una característica que se introdujo en Monero tres años después de su inicio llamada RingCT. Esta nueva capa introducía la posibilidad de ocultar la cantidad de divisa que estaba siendo enviado en una transacción.

El objetivo de este documento es analizar el impacto que ha tenido la introducción de RingCT en Monero y encontrar vulnerabilidades en este nuevo paradigma. Para ello, primero hemos creado una blockchain local en un entorno de pruebas controlado. Sobre este entorno hemos desarrollado una aplicación integrada con la blockchain, con una base de datos y lógica para realizar el procesamiento necesario. Hemos creado miles de transacciones (inyección de datos en la blockchain), trabajado con la información producida por estas en la blockchain y extraído un set de datos. Posteriormente, con Machine Learning (ML) se han analizado estos datos para estudiar el anonimato real que ofrece RingCT.

El trabajo se dividirá en cinco capítulos:

En el primer capítulo se realizará una introducción teórica a blockchain, Bitcoin, las motivaciones detrás de la creación de Monero y las vulnerabilidades técnicas y sociales que éstas presentan.

En el segundo capítulo veremos los tipos de redes que existen en Monero y describiremos el escenario necesario para llevar a cabo el análisis. Se presentarán las herramientas que Monero pone a disposición de usuarios y desarrolladores, tales como servidores dedicados, APIs y aplicaciones de línea de comandos.

En el tercer capítulo se describe la aplicación creada para facilitar la integración con la blockchain y las APIs expuestas por Monero.

Además, dicha aplicación dispone de una base de datos para mantener toda la información necesaria y una capa de servicio para realizar el envío de transacciones y el consecuente procesamiento de los datos.

En el cuarto capítulo hacemos uso del software desarrollado para generar una gran cantidad de transacciones de las que conoceremos toda la información, la procesaremos y la almacenaremos en datasets que se usarán como entrada para diferentes algoritmos de machine learning, con el objetivo de

Por último, haremos un cálculo de la viabilidad económica a la hora de explotar las vulnerabilidades encontradas en el anonimato de Monero.

En el quinto capítulo se extraen las conclusiones de estos análisis y del proyecto en general.

## CAPÍTULO 1. BLOCKCHAIN Y MONERO

Blockchain es una estructura de datos descentralizada creada en 2008 por una persona o grupo de carácter anónimo bajo el pseudónimo de Satoshi Nakamoto en [1]. Esta tecnología nace de la idea de realizar transacciones entre diferentes partes entre las cuales no existe confianza sin la necesidad de ninguna institución central. Una blockchain está mantenida por una cantidad ilimitada de usuarios o nodos, los cuales mantienen un registro de todo el historial de transacciones realizados desde el comienzo de esta.

En palabras de los empresarios Don y Alex Tapscott en [2]:

*"Blockchain es una revolución perfectamente comparable a la aparición del ordenador personal, o al desarrollo y popularización de internet. Es posiblemente, uno de los cambios más importantes y fundamentales que vayamos a ver en nuestras vidas, con el potencial de cambiarlo todo"*

Si bien las transacciones monetarias no son el único uso de blockchain en la actualidad, sí que fue el uso que proponía Satoshi Nakamoto en su artículo y el más extendido, y por lo tanto el uso en el que nos centraremos en este documento. Hablaremos de criptomonedas o criptodivisas para hablar de estas monedas digitales cuya base de datos es una blockchain, y de divisas o dinero fiat [3] para hablar de monedas tradicionales como el euro y el dólar.

Algunas de las principales ventajas de las criptomonedas sobre el dinero fiat son las siguientes:

- **Integridad:** Al ser una base de datos descentralizada y pública no es posible la manipulación de los datos, puesto que al hacerlo el resto de usuarios rechazará esta nueva información. Además toda la información añadida depende de la anterior, por lo que cualquier cambio realizado en un punto antiguo de la blockchain modificaría e invalidaría toda la cadena a partir de ese punto. Más adelante se comenta esto de manera más técnica.
- **Control de masa monetaria:** Las monedas fiat habitualmente están controladas por gobiernos y/o bancos centrales. Es común que estas instituciones intenten arreglar problemas económicos imprimiendo más dinero, lo cual crea inflación, devaluando así el dinero que los ciudadanos tengan ahorrado. En blockchain, en cambio, la masa monetaria está controlada y la cantidad exacta de dinero producido en un determinado rango de tiempo es conocida e inalterable. En muchos casos es también limitada.
- **Infalsificable:** Es imposible añadir transacciones fraudulentas en una blockchain, por lo que el receptor de una transacción siempre tendrá la certeza de que ha recibido un valor legítimo.

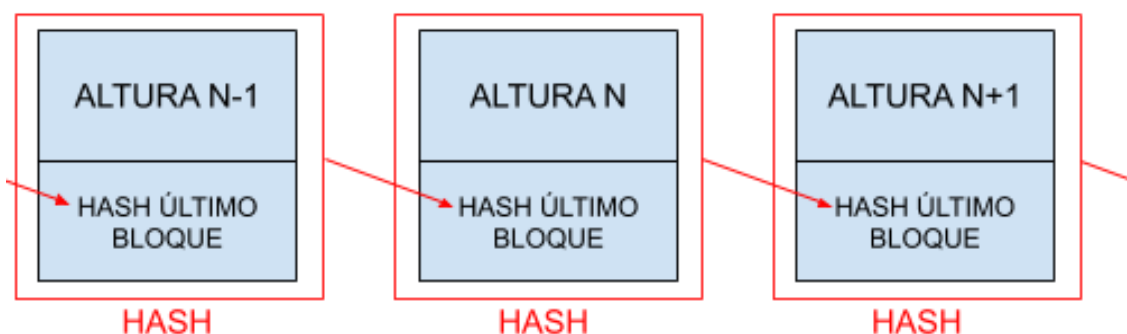
A continuación haremos una introducción más técnica sobre el funcionamiento de una blockchain.

## 1.1. La cadena de bloques

Hemos dicho que blockchain es una cadena de bloques distribuida que es prácticamente inmutable una vez generada. Expliquemos cómo funciona desarrollando el proceso que se sigue desde que un usuario lanza una transacción hasta que ésta acaba contenida en un bloque en la cadena.

Un usuario (llamémosle Bob) tiene una cartera o wallet, que es un software que contiene un par de claves privada y pública, ya que en blockchain generalmente se utiliza criptografía de clave pública. Para más información sobre la criptografía de clave pública ver [4]. La wallet está asociada a una dirección (una cadena de caracteres derivada de la clave pública). Utilizaremos la palabra wallet para referirnos indistintamente al software y a esta dirección. Bob utiliza esta wallet para comunicarse con la red de la blockchain. Bob desea enviarle una transacción a Alice. Para eso necesita conocer la wallet de Alice. Bob lanza la transacción. Entonces, ésta se comunica con máquinas que tienen localmente guardada la blockchain y ejecutan un software (el demonio) que se encarga de sincronizar la blockchain, recibir estas transacciones, verificarlas, etc. Estas máquinas que corren el demonio son conocidas como nodos. Entonces, un usuario que esté interesado en añadir un bloque a la blockchain (estos usuarios son los mineros, y reciben una recompensa por añadir un bloque válido) puede recoger esta transacción y añadirla al bloque que intenta generar. Cuando un bloque es minado correctamente, el minero lo comunica a todos los nodos. Cada nodo acepta el bloque si las transacciones que contiene son válidas. Cuando un nodo acepta un bloque, lo añade al final de su copia de la blockchain.

Hablemos ahora de la integridad de la blockchain. Como comentábamos en el primer apartado, es imposible modificar un bloque ya existente. El principal método que se utiliza en las blockchains para asegurar la integridad e inmutabilidad de un bloque y todos los sucesivos es el hash. Un hash es una función resumen que convierte un valor en otro, tal que el valor original es imposible de obtener a partir del resultante. Cualquier pequeña modificación en el valor original cambia por completo el valor resultante [5]. Cada bloque contiene un hash del bloque anterior, de manera que si cualquier pieza de información dentro de un bloque fuese modificada, ya sea accidental o intencionadamente, el hash de ese bloque y por lo tanto el de todos los bloques siguientes sería modificado, por lo que sería muy sencillo detectar esta modificación.



**Fig. 1.1** Hashes en bloques consecutivos

Si un nodo modifica un bloque y anuncia una nueva rama de blockchain, todos los demás nodos la rechazarán. Es posible que dos mineros anuncien un bloque válido diferente, de manera que unos nodos acepten uno y otros nodos acepten otro, en este caso se puede producir una bifurcación o *fork*, en la que dos versiones diferentes de blockchain conviven. No obstante, unos minutos después cuando un bloque sea minado sobre una de las dos ramas, este se anunciará de nuevo y el resto de nodos aceptarán esa nueva rama como la válida, puesto que los nodos siempre considerarán la rama más larga como la oficial.

Tenemos que hablar también del proceso de minado. Hemos hablado de que encontrar un bloque válido y añadirlo a la blockchain se conoce como minar, pero, ¿qué significa exactamente que un bloque sea válido? ¿Cómo se decide qué minero añade un bloque nuevo? En cada blockchain hay definido un protocolo con unas reglas que debe cumplir un bloque para ser aceptado como válido. Esto se conoce como protocolo de consenso. El más conocido es Proof of Work (PoW), aunque existen otros mecanismos [6].

El objetivo de PoW es encontrar un nonce (un número arbitrario que se utiliza una sola vez) tal que al ser añadido al bloque en cuestión, una función de hash aplicada sobre este bloque produzca un hash que cumpla una cierta característica. Esta característica suele ser que el valor del hash sea inferior a un cierto número, por lo que en la mayoría de blockchains que utilicen PoW como mecanismo de consenso veremos que los hashes de sus bloques empiezan por bastantes cifras a 0. Con esto se consigue que para que un minero pueda conseguir minar un bloque, tenga que tener su máquina generando nonces aleatoriamente, añadiéndolos al bloque y calculando el hash de este bloque, y si aleatoriamente este hash cumple la condición requerida el bloque se considerará válido y por lo tanto minado.

En las blockchains se suele definir un tiempo medio entre bloques, de manera que la introducción de nuevos bloques sea constante y regular. Como la capacidad de cómputo (la llamaremos hashrate a partir de ahora) puede ser muy variable a lo largo del tiempo, para no alterar el tiempo entre bloques añadimos una nueva variable: la dificultad. La dificultad es el parámetro que

marca la condición que tiene que cumplir el hash. La red ajusta automáticamente la dificultad para que con el hashrate actual, la distancia temporal entre bloques sea constante.

Después de esta introducción a las blockchains, pasaremos a explicar la criptomoneda que creó Satoshi Nakamoto como el primer uso de blockchain.

## 1.2. Bitcoin

Satoshi Nakamoto introdujo Bitcoin en el mismo artículo anteriormente mencionado en el que introdujo blockchain [1]. A día de hoy, blockchain y Bitcoin siguen estando estrechamente relacionados, siendo difícil pensar en uno sin pensar automáticamente en el otro. Nakamoto lo define como "Un sistema para transacciones electrónicas sin depender de la confianza".

La blockchain sobre la que funciona Bitcoin utiliza PoW como mecanismo de consenso y su distancia temporal entre bloques es de 10 minutos. Nos adentraremos en su funcionamiento analizando la estructura de sus bloques.


### 1.2.1. Bloque de Bitcoin

Cada bloque de Bitcoin está formado por dos partes: una cabecera y un conjunto de transacciones. En la Fig. 1.2 se puede ver una cabecera de un bloque de Bitcoin, del bloque con altura 628772 extraído de la web [7].

Los campos más relevantes de la cabecera son los siguientes:

- **Hash:** Hash del bloque en cuestión una vez minado con un nonce válido.
- **Confirmations:** Cantidad de nodos que han verificado que el bloque esté correctamente minado.
- **Timestamp:** Marca de tiempo en la que el bloque ha sido minado.
- **Height:** Posición del bloque dentro de la blockchain, empezando por el bloque origen con altura 0.
- **Miner:** Dirección que ha minado el bloque. La web muestra un nombre de usuario conocido, pero lo que está guardado es la dirección del wallet de este usuario.
- **Difficulty:** Es un parámetro que se actualiza en función del hashrate que hay en la red actualmente. Cuanto mayor es la dificultad, más restrictivo es el hash a encontrar para minar un bloque. Se actualiza de manera automática para que cada bloque se tarde una media de 10 minutos en minar en función de ese hashrate.
- **Version:** Versión de Bitcoin en el momento en que se minó el bloque.

- **Hash Merkle:** Es un hash de todos los hashes de las transacciones contenidas en el bloque en cuestión. Es una manera rápida y eficaz de verificar la integridad de las transacciones dentro de un mismo bloque.
- **Nonce:** El número generado aleatoriamente con el que el hash del bloque cumple el requerimiento impuesto según la dificultad.
- **Block Reward:** La cantidad de bitcoin que recibió la wallet que minó el bloque como recompensa.

Hash	000000000000000009df1cc968ef88fa075a314961aa86df03d9b58c10e879 
Confirmations	6
Timestamp	2020-05-03 20:51
Height	628772
Miner	<a href="#">Poolin</a>
Number of Transactions	2,862
Difficulty	15,958,652,328,578.42
Merkle root	ec3757a3d02dc125d4cdee6c13a3f55236cb3f8a6795c97bb846a2c5734617d8
Version	0x20000000
Bits	387,031,859
Weight	3,993,313 WU
Size	1,329,199 bytes
Nonce	540,338,159
Transaction Volume	3488.73190787 BTC
Block Reward	12.50000000 BTC
Fee Reward	0.47181544 BTC

**Fig. 1.2** Cabecera de un bloque de Bitcoin [5]

Si bien en los exploradores gráficos como el que hemos utilizado no se suele mostrar, en la cabecera también existe el **hash del último bloque**, que asegura que en caso de modificar un bloque, todos los bloques posteriores también se vean afectados.

La segunda parte de un bloque contiene sus transacciones. Todos los bloques contienen como mínimo una transacción, llamada transacción coinbase, que es la transacción con la que se recompensa al usuario identificado con la dirección que haya minado el bloque con una cierta cantidad de monedas. A parte de esta transacción encontramos también todas las transacciones que hayan realizado las wallets y se hayan incluido en el bloque. Puede darse el caso en que no existan transacciones disponibles para incluir en el bloque, por lo que un bloque sin transacciones no coinbase también es válido.





## Inputs

HEX ASM

Index	0	Details	Output
Address	39PXg7G9LjNLRsXu3quNxPgPBoynVqKiwr	Value	0.09830796 BTC
Pkscript	OP_HASH160 54725a9c0355f8b5246ea6af6b96a7902e4d93ba OP_EQUAL		
Sigscript	0014a3d4760e8786b11879bb7a3e83082567d14b1cdf		
Witness	304402202657fc662f12a06df02befe001ac9af6648df4664c771446b8e29e4d0b9c54e022042be6ca85cd7dc5fea5a484b656145e901d6266db805077e37eb3819568139ad01 030d85d67dd1de86f620fdf63bcb3dfff1c4bceb45c3e78af80e36588b24dfda9f		

## Outputs

Index	0	Details	Spent
Address	3BxH1nj6YgN38EJxHdRkeB5q3H85wcbXA	Value	0.02837100 BTC
Pkscript	OP_HASH160 70941818cc2c14149fc63d6f20db1b6e73bf406b OP_EQUAL		
Index	1	Details	Spent
Address	3FL9pt7ETzkiAb6zZEHjsuwFcuWrCDeZ6b	Value	0.01664000 BTC
Pkscript	OP_HASH160 959f9f528026fe64c1e3368f306004851e1e56ed OP_EQUAL		
Index	2	Details	Unspent
Address	3F53gE3xxwS9w9C5qwJRjdFx5CYcUcTDPY	Value	0.05290649 BTC
Pkscript	OP_HASH160 92c43ff29916533cc734803b467578fe00a33e67 OP_EQUAL		

**Fig. 1.4** Inputs y outputs en una transacción de Bitcoin [7]

En una transacción es común dividir un input entre varios outputs para enviar diferentes cantidades a diferentes wallets. Es también común que la cantidad de dinero a enviar no coincida con el valor de un input disponible, por lo que se puede crear un output hacia la wallet del remitente con la diferencia, lo que coloquialmente conocemos como cambio.

Sumando el valor de los tres outputs de la transacción mostrada en la Fig. 1.3 y detallados en la Fig. 1.4, más el valor de la comisión de la transacción obtenemos el valor del input:

$$0.02837100 + 0.01664000 + 0.05290649 + 0.00039047 = 0.09830796$$

Y si sumamos todas las comisiones de las transferencias dentro de un mismo bloque obtendríamos el valor de la recompensa de comisiones mostrado en la cabecera.

Hasta aquí hemos visto a grandes rasgos cómo funciona Bitcoin y sus ventajas sobre el dinero fiat. Ahora bien, no todo en Bitcoin es positivo. A continuación veremos algunos de sus mayores inconvenientes.

## 1.2.2. Problemas de Bitcoin

Dividiremos los inconvenientes más relevantes de Bitcoin entre técnicos y sociales.

### 1.2.2.1. Problemas técnicos

Uno de los problemas más comunes en Bitcoin es su gasto energético tan extremadamente alto. Todo nodo que haya trabajado para encontrar un nonce y no lo haya conseguido, ha desperdiciado toda la electricidad utilizada. Sin embargo, esto es un problema común a cualquier blockchain que utilice PoW como mecanismo de consenso.

Otro problema que sí es especialmente grave en el caso de Bitcoin es el minado por hardware. Bitcoin utiliza Hashcash como algoritmo de minado [8]. En las primeras etapas de Bitcoin, los mineros de Bitcoin utilizaban sus ordenadores para minar, por lo que sus CPUs eran las encargadas de ejecutar este algoritmo para encontrar nonces válidos. Sin embargo, un tiempo más tarde otros usuarios descubrieron que con sus tarjetas gráficas (GPU) podían minar Bitcoin unas cien veces mejor que con una CPU (es decir, se podían probar unas cien veces más nonces que con un CPU). Si bien los usuarios que minaban con sus CPU aún podían competir con aquellos que utilizaban GPUs, sus ganancias provenientes de recompensas por minar bloques se vieron muy reducidas. El problema real llegó con la aparición de hardware especial para el minado de Bitcoin. Este hardware, conocido como ASIC (Application-Specific Integrated Circuit) es capaz de minar unas seis órdenes de magnitud más rápido que una CPU, por lo que a partir de la llegada de estos circuitos se hizo completamente inviable minar con GPU o CPU, ya que el coste en electricidad necesario para minar un Bitcoin era muy superior al valor de éste. A día de hoy, en 2020, la inmensa mayoría de bloques minados de Bitcoin provienen de granjas de minado, unos equipos especializados con decenas de equipamiento ASIC en países con la electricidad muy barata como China o India. Recordemos que uno de los principios básicos de la existencia de Bitcoin era su descentralización. La minería ASIC, en cambio, ha vuelto a centralizar el minado en un grupo relativamente reducido de mineros que controlan la mayoría de estas granjas [9].

### 1.2.2.2. Problemas sociales

Imaginemos que siempre que deseásemos realizar una compra debiésemos dar a conocer todo nuestro historial de transacciones, revelando nuestro balance total de dinero en cada momento de nuestra vida, cada persona, empresa y organización a la que hemos pagado o de la que hemos recibido

dinero, qué cantidad y en qué momento. Pensemos en toda la cantidad de implicaciones que tiene esto:

- Alicia va a la panadería que hay en el barrio al que se acaba de mudar. No se lo ha dicho a nadie, pero es millonaria. Cuando compra el pan, el panadero automáticamente debe revisar su cuenta, por lo que se da cuenta de su poder adquisitivo. Al día siguiente, Alicia vuelve a la panadería a la misma hora. Al tercer día vuelve, y ahora hay dos amigos del panadero esperándola armados para extorsionarla a cambio de un porcentaje de su riqueza. Alicia jamás podrá comprar nada sin exponerse.
- Carlos actualmente cobra 22 bitcoins anuales en su empleo. Quiere cambiar de empresa puesto que sabe que su empleo los sueldos son mucho mayores. Él pide 38 bitcoins, pero desde la nueva empresa observan que antes solo cobraba 22000 bitcoin, por lo que se niegan a pagarle prácticamente el doble.
- María va a la universidad y normalmente a las 11:30 compra un café en la cafetería. Por las tardes va al gimnasio, al cual paga una cuota mensual, pero alguna vez también compra una bebida isotónica en la máquina expendedora. Los jueves va al supermercado a hacer la compra de comida semanal. Los sábados sale con sus amigas a un bar de copas de su barrio. María empieza a ser acosada por un hombre, que conociendo las wallets de todos estos establecimientos que ella suele visitar, ha podido trazar su rutina y saber aproximadamente donde está en cada momento.

Estas son solo algunas de las implicaciones que conlleva la falta de anonimato de Bitcoin. Si pretendemos que las criptodivisas tomen un papel relevante en la sociedad necesitamos un nivel de anonimato mucho mayor.

A raíz de estos inconvenientes de Bitcoin la comunidad comenzó a desarrollar algunas alternativas con el propósito de solucionar estos problemas. Una de estas criptodivisas alternativas muy interesante es Monero.

### 1.3. Monero

Monero es una criptomoneda que enfatiza la privacidad y anonimato de sus usuarios. Fue creada en 2014 como fork (ramificación del código fuente) de otra criptomoneda, Bytecoin (a pesar de que el nombre pueda sugerir alguna relación con Bitcoin, no tienen nada que ver), a la cual superaría rápidamente en popularidad [5].

A lo largo de todo el documento hablaremos de Monero (con la primera letra en mayúscula) para referirnos a la divisa, monero (con la primera letra en minúscula) para referirnos a una moneda de esta divisa y moneroj como plural

de monero. Monero tiene hasta 12 posiciones decimales, y el nombre de estas unidades se forma de la manera: prefijo decimal + "nero". Por ejemplo,  $10^{-3}$  equivaldría a 1 milinero. Nosotros utilizaremos en diversas ocasiones  $10^{-12}$  como piconero. 1 piconero es la cantidad mínima divisible de Monero, y también se le conoce como "unidad atómica". Como la mayoría de criptomonedas tiene un acrónimo asociado: XMR.

El formato de las direcciones de Monero se explica en el capítulo 2.

La cabecera de un bloque de Monero tiene una estructura muy similar al de Bitcoin.

↑ Height	2090392
🕒 Timestamp	2020-05-03 18:20:07 UTC
📊 Block difficulty	154817842280
↔ Block size (bytes)	97158
↔ Cumulative Difficulty	50590527449564828
↔ Total Generated Coins	17546565.000065129250
⇄ Transactions	36

**Fig. 1.5** Cabecera de un bloque de Monero [10]

En cambio, como veremos más adelante, en las transacciones hay diferencias muy significativas. Para entender estas diferencias, explicaremos cuatro de las ventajas que tiene Monero sobre Bitcoin.

### 1.3.1. Características de Monero

#### 1.3.1.1. Cuatro claves en lugar de dos

Bitcoin utiliza un par de claves pública y privada para cifrar sus transacciones. En cambio, en Monero se utilizan dos pares: un par de visualización y otro par de gasto. En el anexo A se explica la generación de una transacción con dos pares de claves de manera matemática.

Este sistema proporciona más anonimato y más versatilidad que el único par de claves clásico. En realidad, podemos decir que en el sistema de único par

de claves existen dos roles posibles respecto a una dirección, dependiendo de si se conoce la clave privada o no: si se conoce es posible gastar los bitcoin de la wallet, si no se conoce únicamente es posible visualizar los movimientos que se realicen desde esta. En Monero, en cambio, como se usan dos claves privadas pueden haber hasta cuatro roles distintos relacionados con una dirección, según si se conoce solo la clave privada de gasto, solo la de visualización, ambas o ninguna. Un ejemplo puede ser el siguiente: Charlie es una organización benéfica que requiere máxima transparencia con el dinero que maneja. Para esto, podría hacer pública su clave privada de visualización por lo que cualquiera podría comprobar la cantidad de fondos de los que dispone, pero en ningún caso podrían ser robados ya que para ello también se necesitaría la clave privada de gasto [5].

#### 1.3.1.2. Firma en anillo

Monero utiliza la firma en anillo como medida de protección de la privacidad de sus usuarios. En términos generales, la firma en anillo es un tipo de firma digital en la que la firma se ofusca dentro de un grupo de  $k$  firmantes empleando la clave pública de cada uno de los miembros. Es posible verificar que la firma ha sido realizada por uno de los miembros del anillo, pero es matemáticamente imposible saber exactamente qué miembro.

En el caso de Monero se aplica una firma en anillo en las transacciones. Como se explica en el capítulo 1.2.1, en Bitcoin una transacción se compone de uno o más inputs que referencian a un output anterior recibido por la misma wallet. En Monero, en cambio, un input no referencia un output directamente, sino que este input se muestra en forma de firma en anillo. Este anillo contiene el output real y otros inputs que pueden pertenecer o no a la misma wallet. A todos los outputs a excepción del real lo nombraremos como señuelo o *decoy*.

Para prevenir el doble gasto del mismo output se introducen las *Key Image (KI)*. Una KI es una clave criptográfica derivada del output utilizando la clave privada de gasto del emisor. De esta manera, como el mismo output siempre produciría la misma KI es sencillo comprobar si esa KI ya existe en algún anillo anterior.

En el ejemplo de la Fig. 1.6 se puede ver un anillo contenido en una transacción anteriormente mostrada en la Fig. 1.5. En este anillo se están referenciando 11 outputs diferentes de transacciones antiguas. Solo uno de esos outputs produce la KI asignada al anillo. El resto son los señuelos.

Inputs (1)		
	Amount	Key Image
—	0.000000000000	288ac1d07686f8fe2bc9b02e1ccc48a4a60fb289d3e8eae7c614cb4634b6ebf2
From Block		Public Key
2083703		6f9452f23f76d14dba2256d10d8285401ba89cca9658f0e68d95ab7b82e8f725
2084547		6a97d14531643d2d00feac943d7e995d72fca19ea88154568097dba874b7ab8f
2084811		07a355fbcd09874fd1692f101890105d933a6a2004e5db0749775689fb6bba77
2085042		36c3c1d980468a5f56ed71b7f500952259cf7ca47d9031741f39ea8939fdff6c5
2085635		793b5758e5df7a962069c888ee6ff8a7cb669f5705e9fb473ca7946e26685d0
2087894		f2b5b36a065e8b4c3c4ec9e58eeefed9c57f807190dccc784850113dd9773ff9
2089662		3e863826db7aaad55d68b9bde664caa3c023d8366ad48bde5a757bb77e362051
2089850		1f81da8f61f9616e27c0272c007b29d148d819defa44adc33302f13e81a5194e
2089868		7a8ec92cd949e3d80e84b18a6b625c2d1f4138ed78dfb3887e321daf3a0494fb
2090058		d5d692b23af1e1c2d1a6b9dab2ed5f357099529f3150040373b4ac88df783d80
2090091		e37257c40d70494b224e4889415ac739e982a9649c5e78962b7c15027f84c05f

**Fig. 1.6** Anillo en transacción de Monero [10]

De esta manera, como un mismo output siempre producirá la misma KI, en la verificación de una transacción, el nodo simplemente tiene que mirar si esa KI ya existe previamente en la blockchain. En caso afirmativo, ese output ya ha sido gastado anteriormente y la transacción debe ser rechazada [11].

### 1.3.1.3. Subdirecciones

La motivación principal para esta característica es el hecho de poder tener diferentes direcciones en las cuales recibir sin que sean relacionables. Una subdirección es una dirección secundaria derivada de una principal. Desde la misma wallet abierta en línea de comandos se pueden gestionar todas las direcciones, tanto la principal como todas sus subdirecciones. Una subdirección se genera de manera criptográfica utilizando las claves privadas de la dirección principal, por lo que a ojos de cualquier otro usuario es imposible obtener la dirección principal o cualquier otra subdirección a partir de una subdirección conocida.

David tiene una dirección de Monero pública en la que por ejemplo su empleador le paga la nómina (actualmente en el año 2020 hay empresas que lo hacen). Pero también está trabajando en un proyecto para el que necesita financiación pero no quiere revelar su identidad. Entonces puede crear una subdirección y publicarla en la web de este proyecto. Las donaciones aquí recibidas podrán ser gestionadas desde la wallet de su dirección principal y siempre que lo necesite podrá transferir los fondos entre ellas.

Otra ventaja de las subdirecciones es la divisibilidad de pagadores. Pongamos que David, además de pedir financiación desde la web del proyecto, lo hace en varios foros y redes sociales. En este caso podría crear una subdirección para cada lugar en el que quiera publicitarse, y de esta manera tener controlado fácilmente cuanto ha recibido en cada wallet, y así analizar el nivel de implicación de cada web con su causa [12].

La creación de una subdirección se explica en el capítulo 2.2.

#### 1.3.1.4. RingCT

Ring Confidential Transactions (RingCT de ahora en adelante) es un método con el cual las cantidades de los outputs están ocultas. RingCT fue introducido en enero de 2017 y desde septiembre del mismo año fue obligatorio utilizarlo en sus transacciones. A partir de ese momento, todos los outputs tenían sus cantidades escondidas excepto aquellos generados por las transacciones coinbase, ya que la recompensa por minar un bloque es conocida.

Cada output no oculto con RingCT se define como un par de valores cantidad e índice, es decir que el primer output de cantidad  $n$  se puede denominar como output  $(n, 1)$ , el segundo de esa misma cantidad  $(n, 2)$ , etcétera. En cambio, a partir del primer output que utiliza RingCT, como las cantidades pasan a ser ocultas los outputs pasan a definirse simplemente por su índice, por lo que podemos hablar del output 1, output 2, etc. Evidentemente, esta numeración es única sin hacer distinción de cantidades [13].

En las Fig. 1.7 y 1.8 se puede ver la diferencia entre una transacción coinbase (y por lo tanto sin RingCT) y una transacción normal con RingCT, respectivamente. En la primera se muestran las cantidades de los outputs mientras que en la segunda no.

Transaction `ca72d95746a0c5f9eda822f28f82911df32d062cab873cb07937d61fc23b53e`

From Block	2090392
Output total	1.738568790240 XMR
Fee	0.000000000000 XMR
Size	90 bytes
Mixin	0
Unlock	2090452

Outputs (1)	
Amount	Public Key
1.738568790240	124dc8aa6c5becd29f4bcd62720fb10fb9dff773c666844c6c5fc9a40a76410a

Fig. 1.7 Transacción coinbase de Monero [9]



Transaction 67bb544eb749a0aef2134309a5446a1a8fc1521285a24f59a989b864345b7e5f

From Block	2090392
Output total	confidential
Fee	0.020293760000 XMR
Size	1773 bytes
Mixin	10
Unlock	0

Confidential Transaction — amounts are not disclosed.

Inputs (1)	
Amount	Key Image
+	0.000000000000
	288ac1d07686f8e2bc9b02e1ccc48a4a60fb289d3e8eae7c614cb4634b6ebf2
Outputs (2)	
Amount	Public Key
0.000000000000	133c45da3cf9c92b49d144442dc8903b37c7373cef549f9ff44e8bd04ee5bdd
0.000000000000	d545f5d4645774fa16f1a3ecca27d64b975b0a4d8e4488f5b39375e220fc3d

**Fig. 1.8** Transacción no coinbase de Monero [10]

En el tercer punto del apartado siguiente se explica con más detalles las ventajas e inconvenientes que supone la introducción de RingCT, para más adelante realizar todo un análisis práctico de una posible manera de obtener el output real en un anillo en la blockchain post-RingCT.

### 1.3.2. Vulnerabilidades de Monero

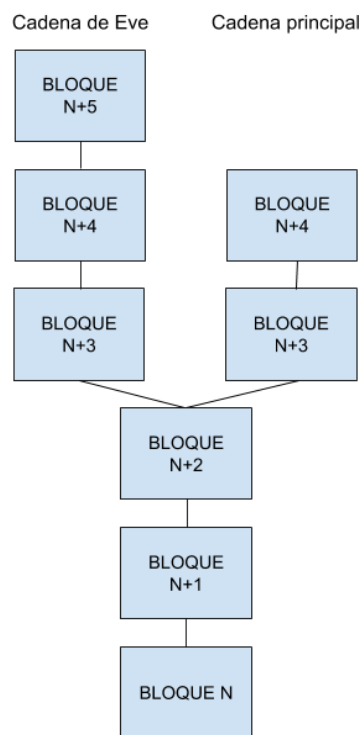
En este apartado haremos un repaso de las vulnerabilidades conocidas de Monero más relevantes.

#### 1.3.2.1. Ataque del 51%

Realmente este ataque no es exclusivo en Monero sino en cualquier criptomoneda que utilice Proof of Work (incluyendo Bitcoin) como mecanismo de consenso, pero debemos comentarlo debido a su relevancia.

Este ataque se basa en la posibilidad de que, si alguien puede controlar más del 50% del poder computacional de minería de la red, puede tener el control absoluto de la misma, y gastar dos veces la misma cantidad de divisa.

Pongamos por ejemplo que Eve hace un pedido online a Alicia y le paga con Monero. En el momento en que realiza el pedido la blockchain tiene una altura N. Esta transacción podrá ser validada en cualquier bloque a continuación de éste. Pongamos que, por ejemplo, la transacción se termina de validar en el bloque N+3. A partir de este bloque, en la cadena oficial, Eve ya no dispondrá de esos fondos. Ahora bien, si Eve controla el 51% o más de hashrate de la red, podrá comenzar a minar bloques a partir del N+2. Como en la blockchain ya hay un bloque a continuación de éste, en principio no se considerará oficial ya que esta cadena alternativa, o fork, es más corto. Pero tarde o temprano, Eve, al poder minar bloques de manera más frecuente que el resto de la red, conseguirá que su fork tenga más bloques que la cadena oficial. En este momento, la cadena oficial pasará a ser la suya. El bloque N+3 en el que se había validado su envío de fondos a Alicia ya no formará parte de la cadena oficial, por lo que Eve nunca ha gastado su dinero, pero ya habrá recibido el pedido de parte de Alicia.



**Fig. 1.9** Fork producido por un ataque del 51%

#### 1.3.2.2. *Cryptojacking*

Tal y como se comentaba en el capítulo anterior, Monero tiene diferentes ventajas sobre Bitcoin, las cuales se pueden resumir en dos puntos:

- Fungibilidad gracias a mecanismos de criptografía avanzada. La fungibilidad monetariamente hablando significa que cada moneda es exactamente igual de válida sin tener en cuenta su procedencia.
- Bajas posibilidades de centralización debido a la resistencia al minado por hardware.

Si bien estos dos puntos pueden parecer muy positivos por separado, el hecho de encontrarlos juntos en la misma criptomoneda presenta un inconveniente. Imaginemos que Alice es la administradora de una página web con una cantidad de visitas considerable. Alice decide rentabilizar al máximo su web y añade un script en código Javascript a su web que será cargado en cualquier navegador que acceda a esta web. Este script se encargará de minar bloques en la red de Monero para la wallet de Alice. Recordemos que Monero utiliza Proof of Work como protocolo de consenso, por lo que el cliente que descargue esta web en su dispositivo, pondrá un porcentaje elevado del poder computacional de su CPU y/o GPU al minado. Esto incurrirá en un mayor gasto energético, y, en última instancia, una mayor factura energética. Por otra parte, Alice estará recibiendo moneroj en su wallet de las comisiones y de las transacciones de base que han minado los visitantes a su web. Estos moneroj lo puede utilizar como moneda en sí misma o bien intercambiándolo por otras criptomonedas o dinero fiat, pero en cualquier caso utilizarlo como dinero. En otras palabras, Alice estará robando a cualquier persona que cargue su script en uno de sus dispositivos. Este tipo de ataque es conocido como cryptojacking.

Ahora bien, ¿por qué esta práctica solo se ha llevado a cabo en Monero, y no en otras criptomonedas más populares? Veamos qué pasaría si Monero no cumpliera las dos características explicadas en el párrafo anterior, como es el caso de la mayoría de criptomonedas.

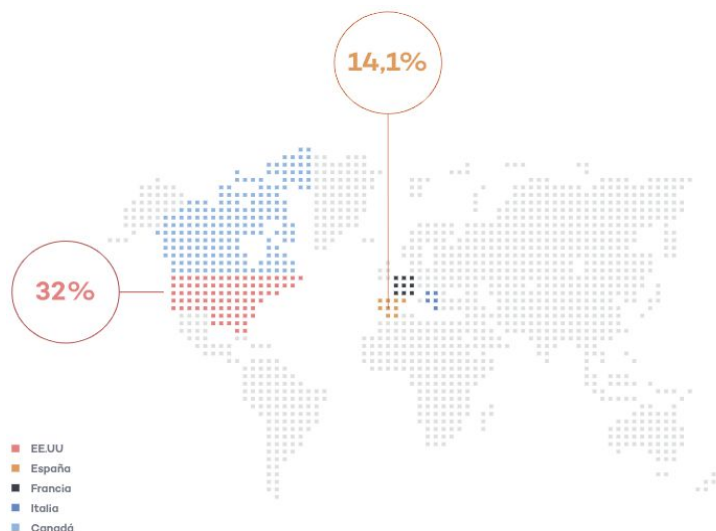
¿En qué afecta la fungibilidad?

Imaginemos que Alice hace que sus víctimas minen Bitcoin para ella. En este caso, su wallet debería estar expuesta. Cuando las primeras víctimas se diesen cuenta de lo que está ocurriendo, simplemente habría que mirar en la blockchain qué bitcoins han sido minados por Alice. Sería posible hacer pública una lista de todos los outputs que tengan como destinatario su wallet, y que a partir de ese momento, cualquier bitcoin que hubiese sido creado como recompensa por minar un bloque por parte de una víctima, o bien que hubiese sido incluido como comisión en uno de estos bloques, fuese automáticamente rechazado por el minero. De esta manera se desincentivaría esta práctica. En cambio, en el caso de Monero, cada una de las monedas es completamente fungible y es imposible saber si en algún momento anterior ha pasado por las manos de Alice.

### ¿En qué afecta la resistencia ASIC?

Ya hemos visto que a día de hoy es completamente inviable minar Bitcoin con ordenadores ni con cualquier otro dispositivo que necesite ejecutar software para minar. Si Alice forzase a sus víctimas a minar Bitcoin, éstas estarían compitiendo por encontrar nonces válidos contra granjas de minado con millones de veces más hashrate. Sus beneficios serían prácticamente nulos. No merecería la pena. En cambio, al minar Monero, este dispositivo estaría compitiendo contra otros dispositivos similares, por lo que Alice si puede obtener cierta rentabilidad.

Según un informe de Panda Adaptive Defense 360 [14], en 2017 fueron minados a través de esta técnica una cantidad de criptomonedas cuyo valor al cambio en ese momento alcanzaba los 250.000\$ mensuales. Cabe también destacar que España fue el segundo país del mundo con mayor cantidad de webs infectadas (14.1%), solo por detrás de Estados Unidos (32%).



**Fig. 1.10** Geografía del cryptojacking [14]

El primer y más grande servicio de minado de criptomonedas fue Coinhive. El modelo de negocio de esta empresa era el siguiente: Ofrecían a terceros una integración con su plataforma en la cual se podían registrar de manera tradicional con un usuario y contraseña. Coinhive gestionaba las wallets por sus usuarios. Estos usuarios podían insertar un simple script proporcionado por Coinhive, de forma que cada vez que su página web fuese cargada en un navegador, éste comenzaría a minar Monero para ellos. Coinhive a cambio se quedaba con el 30% de las ganancias de sus clientes.

Coinhive tuvo que dejar de operar debido a que el aumento de hashrate de la red hizo que su modelo dejase de ser rentable.

El primer objetivo de este proyecto fue la creación de un método para proteger a los usuarios del cryptojacking. No obstante, a día de hoy, el cryptojacking

está resuelto casi en su totalidad mediante software de seguridad, como antivirus, que pueden detectar si el dispositivo en el que se ejecutan está minando. También algunos navegadores, especialmente Google Chrome, han creado mecanismos para detectar y evitar cualquier script malicioso. El filtro CORS también se encarga de evitar que un script descargado desde un dominio haga llamadas a otro dominio. Por estos y otros motivos, en la parte práctica del proyecto nos centraremos en el ataque que veremos a continuación.

#### 1.3.2.3. Trazabilidad antes y después de RingCT

En [15] los autores realizan un análisis de la trazabilidad de los outputs, con el objetivo de obtener la wallet de la que proviene una transacción. Principalmente aprovechan dos debilidades:

- Muchas transacciones anteriores a febrero de 2017 se lanzaban sin señuelos, por lo que cada de input se deduce automáticamente su output asociado. Actualmente esta vulnerabilidad está solucionada puesto que los nodos no verifican una transacción si no se ha escogido una cantidad de señuelos de como mínimo 10.
- Antes de RingCT, como las cantidades de los outputs eran públicas, todos los outputs que formasen un anillo debían contener la misma cantidad de moneroj. Tengamos en cuenta que toda la divisa generada proviene de transacciones coinbase, cuyo valor es diferente en cada bloque, con cantidades con muchas cifras y poco frecuentes, como "14054093900706" o "27225879557239". A la hora de buscar señuelos para un output con estas cantidades, la distribución de estos señuelos quedaba de manera antinatural, con unos patrones que no se corresponden con la actividad humana. Analizando estos patrones se detecta que con mucha frecuencia (estiman en el 80.35% de los casos), el output real era el más reciente puesto que en muchas ocasiones se debía ir muy hacia atrás en la blockchain para encontrar señuelos con esas cantidades tan infrecuentes.

La firma en anillo es una de las piezas clave que otorgan a Monero un anonimato sin precedentes en blockchain. Sin embargo, si bien la recuperación del firmante real es matemáticamente imposible, puede ser posible hacerlo mediante otros métodos. Si podemos generar suficientes inputs como para que en un determinado output de origen desconocido, todos los inputs referenciados excepto uno sean nuestros, podremos reconocer cuál de ellos es el real. Es decir, para un anillo típico actual con  $k = 11$ , si logramos reconocer como falsos 10 de los outputs automáticamente reconoceremos el real.

En los capítulos siguientes construiremos un entorno y desarrollaremos una aplicación con tal de realizar este ataque y analizar su viabilidad.

## CAPÍTULO 2. ENTORNO EN MONERO TESTNET

En este apartado construiremos una blockchain privada de Monero. En ella tendremos nodos, cada uno con una copia de la blockchain, y wallets que se conectarán a estos nodos. Esto nos permitirá hacer cualquier prueba que deseemos con total control y sin ningún riesgo económico con el objetivo de familiarizarnos con el funcionamiento.

Por motivos evidentes, como la dificultad para minar un bloque, el tiempo entre minado de bloques, el alto descontrol sobre transacciones y muchos más, es inviable realizar nuestro escenario sobre la red principal de Monero. Necesitaremos una red de pruebas. Monero ofrece tres tipos de red:

- **Mainnet:** Es la red oficial, la única en la que los moneroj tienen valor real.
- **Stagenet:** Es una red idéntica a la mainnet que puede ser utilizada para realizar pruebas de desarrollo antes de pasar el código a producción, o para familiarizarse con ésta antes de pasar a la oficial.
- **Testnet:** Es una red de pruebas para desarrolladores.

En momentos en los que se necesite especificar, hablaremos de *main blockchain*, *stage blockchain* y *test blockchain* para referirnos a cada una, respectivamente. Cada red es completamente independiente de las otras y no es posible mover ningún tipo de información entre ellas. Siempre que necesitemos utilizar la main blockchain lo haremos a través de uno de sus nodos remotos oficiales con URL <http://node.moneroworld.com:18089>.

Debemos hablar también del formato de las direcciones. Una dirección de Monero ocupa 69 bytes, cuyo formato es:

**Byte 0:** Identifica el tipo de red y el tipo de dirección (Network Byte).

**Bytes 1-32:** Clave pública de gasto.

**Bytes 33-64:** Clave pública de visualización.

**Bytes 65-68:** Checksum.

	Mainnet	Stagenet	Testnet
Main Address	4	5	9
Subaddress	8	7	B

Utilizaremos una testnet en la que podremos tener nuestra propia test blockchain, controlar todos los parámetros e incluso utilizar código fuente

modificado de Monero. Por lo tanto todas nuestras direcciones empezarán por 9 o B, según si sean direcciones principales o subdirecciones.

Para empezar a montar nuestro escenario, descargaremos el código fuente desde el repositorio oficial en GitHub en el siguiente enlace: <https://github.com/monero-project/monero>. El sistema operativo con el que trabajaremos es Ubuntu 18.04.4 LTS en un ordenador con 8 GB de RAM y un disco duro de 500 GB.

Seguimos las instrucciones para compilar el código en Linux. Una vez finalizadas, tendremos creados una serie de binarios en el directorio `~/Developer/monero/build/Linux/master/release/bin`. Nos interesan dos de ellos, el demonio y la wallet CLI:

- `./monerod` (*monero daemon*)
- `./monero-wallet-cli` (*monero wallet command line interpreter*)
- `./monero-wallet-rpc` (*monero wallet remote procedure call*)

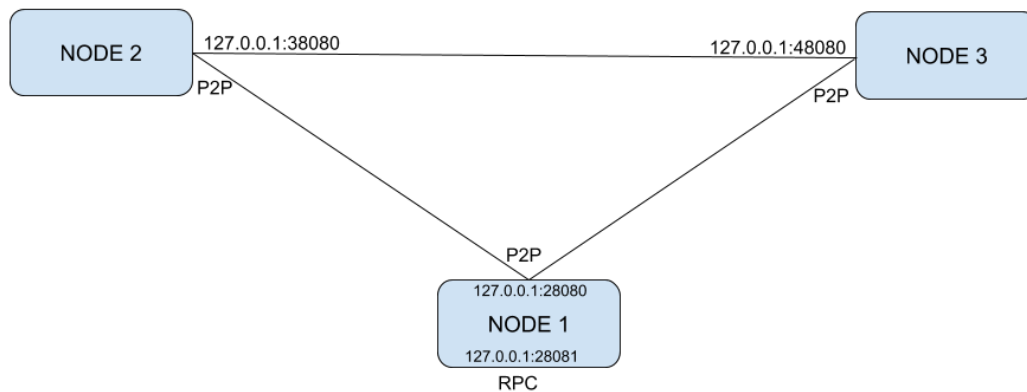
A continuación veremos detalladamente el funcionamiento de cada uno y qué uso le daremos.

## 2.1. Monero Daemon

Recordemos que la main blockchain de Monero actualmente ocupa unos 78 GB y este tamaño está en constante crecimiento. Además, para sincronizar la blockchain no es aconsejable descargarla sin ningún tipo de verificación. Es aconsejable utilizar el demonio para sincronizarla y verificar que todos los bloques son correctos.

Teniendo esto en cuenta, es inviable para muchos usuarios tener la blockchain descargada localmente, sobretodo si utilizan wallets en dispositivos móviles. Aquí entran en juego los nodos remotos. Un nodo remoto es un servidor dedicado con el que cualquier wallet puede sincronizarse y delegar en él cualquier operación que requiera lectura o escritura sobre la blockchain. En estos nodos se ejecuta el demonio de monero [16], el cual dispone de una API RPC. Esta API será la manera en que los usuarios interactuarán con el nodo.

En la Fig. 2.1 se muestra un esquema del escenario que tendremos inicialmente. Tendremos tres nodos corriendo en la misma máquina, cada uno en un puerto diferente.



**Fig. 2.1** Escenario con tres nodos conectados

El comando que utilizaremos para arrancar el demonio es el siguiente:

```
./monerod --testnet --p2p-bind-port 28080 --p2p-bind-ip 0.0.0.0 --rpc-bind-port 28081 --zmq-rpc-bind-port 28082 --no-igd --hide-my-port --data-dir ~/testnet/node_01 --p2p-bind-ip 127.0.0.1 --log-level 4 --add-exclusive-node 127.0.0.1:38080 --add-exclusive-node 127.0.0.1:48080 --fixed-difficulty 3000 --log-file ~/testnet/logs/node01.log --confirm-external-bind
```

Con el primer parámetro se especifica que estamos utilizando una testnet. En el anexo B se desglosa el comando con todos los parámetros.

En la Fig. 2.2 vemos el inicio del tercer nodo esperando conexiones p2p en el puerto 48081, cuando los nodos en los puertos 28080 y 38080 ya están activos y escuchando. Copiaremos el comando descrito anteriormente dos veces más, uno para cada nodo del escenario. Los llamaremos nodo 1, nodo 2 y nodo 3, en los puertos 28080, 38080 y 48080 respectivamente. En la imagen, el nodo 3 tiene el nivel de los logs a 1 (información).

Cada nodo tiene su propia copia de la blockchain en su directorio, y estos se sincronizarán entre ellos con tal de compartir los bloques y la pool (una base de datos en memoria en cada demonio en la que se guardan las transacciones recibidas que todavía no han sido minadas).



```

samuel@samuel-msi:~/testnet$ ./start_node03.sh
2019-12-16 11:26:28.767 I Monero 'Boron Butterfly' (v0.14.1.2-b60cf6a93)
2019-12-16 11:26:28.767 I Moving from main() into the daemonize now.
2019-12-16 11:26:28.767 I Initializing cryptonote protocol...
2019-12-16 11:26:28.767 I Cryptonote protocol initialized OK
2019-12-16 11:26:28.768 I Initializing p2p server...
2019-12-16 11:26:28.768 I Setting LIMIT: 2048 kbps
2019-12-16 11:26:28.768 I Set limit-up to 2048 kB/s
2019-12-16 11:26:28.768 I Setting LIMIT: 8192 kbps
2019-12-16 11:26:28.768 I Setting LIMIT: 8192 kbps
2019-12-16 11:26:28.768 I Set limit-down to 8192 kB/s
2019-12-16 11:26:28.768 I Setting LIMIT: 2048 kbps
2019-12-16 11:26:28.768 I Set limit-up to 2048 kB/s
2019-12-16 11:26:28.768 I Setting LIMIT: 8192 kbps
2019-12-16 11:26:28.768 I Setting LIMIT: 8192 kbps
2019-12-16 11:26:28.768 I Set limit-down to 8192 kB/s
2019-12-16 11:26:28.768 I Resolving node address: host=163.172.182.165, port=28080
2019-12-16 11:26:28.768 I Added node: 163.172.182.165:28080
2019-12-16 11:26:28.768 I Resolving node address: host=195.154.123.123, port=28080
2019-12-16 11:26:28.768 I Added node: 195.154.123.123:28080
2019-12-16 11:26:28.768 I Resolving node address: host=212.83.172.165, port=28080
2019-12-16 11:26:28.768 I Added node: 212.83.172.165:28080
2019-12-16 11:26:28.768 I Resolving node address: host=212.83.175.67, port=28080
2019-12-16 11:26:28.768 I Added node: 212.83.175.67:28080
2019-12-16 11:26:28.768 I Resolving node address: host=5.9.100.248, port=28080
2019-12-16 11:26:28.768 I Added node: 5.9.100.248:28080
2019-12-16 11:26:28.769 I Set server type to: 2 from name: P2P, prefix_name = P2P
2019-12-16 11:26:28.769 I Binding (IPv4) on 127.0.0.1:48080
2019-12-16 11:26:28.769 I Net service bound (IPv4) to 127.0.0.1:48080
2019-12-16 11:26:28.769 I p2p server initialized OK
2019-12-16 11:26:28.769 I Initializing core RPC server...
2019-12-16 11:26:28.769 I Set server type to: 1 from name: RPC, prefix_name = RPC
2019-12-16 11:26:28.769 I Binding on 127.0.0.1 (IPv4):48081
2019-12-16 11:26:28.769 I Generating SSL certificate
2019-12-16 11:26:30.205 I core RPC server initialized OK on port: 48081
2019-12-16 11:26:30.205 I Initializing core...
2019-12-16 11:26:30.205 I Loading blockchain from folder /home/samuel/testnet/node_03/testnet/lmdb ...
2019-12-16 11:26:30.205 W The blockchain is on a rotating drive: this will be very slow, use an SSD if possible
2019-12-16 11:26:30.208 I batch transaction mode already enabled, but asked to enable batch mode
2019-12-16 11:26:30.208 I batch transactions enabled
2019-12-16 11:26:30.208 I Blockchain initialized. last block: 1127, d39.h0.m56.s9 time ago, current difficulty: 3000
2019-12-16 11:26:30.208 I Loading checkpoints
2019-12-16 11:26:30.208 I Core initialized OK
2019-12-16 11:26:30.208 I Starting core RPC server...
2019-12-16 11:26:30.208 I Run net_service loop( 2 threads)...
2019-12-16 11:26:30.208 I core RPC server started ok
2019-12-16 11:26:30.210 I Starting ZMQ server...
2019-12-16 11:26:30.210 I ZMQ server started at 127.0.0.1:48082.
2019-12-16 11:26:30.210 I Starting p2p net loop...
2019-12-16 11:26:30.210 I Run net_service loop( 10 threads)...
2019-12-16 11:26:30.211 I [127.0.0.1:57674 40ec91ff-54f1-40cf-ab39-d33cc603acd4 INC] NEW CONNECTION
2019-12-16 11:26:30.211 I
2019-12-16 11:26:30.211 I *****
2019-12-16 11:26:30.211 I You are now synchronized with the network. You may now start monero-wallet-cli.
2019-12-16 11:26:30.211 I

```

Fig. 2.2 Nodo 3 inicia y sincroniza con sus peers

### 2.1.1. Monero Daemon RPC

La API RPC (Remote Procedure Call) del demonio de Monero define una gran cantidad de métodos. Es una interfaz a través de la que introducir algunos de los comandos anteriormente explicados. La mayoría de los métodos se ejecutan como un HTTP POST, como application/json. Algunos métodos requieren un parámetro llamado "params" que tomará el valor de un objeto javascript cuyos parámetros vendrán definidos en cada método [17].

A continuación se explican los métodos que utilizaremos.

### **get\_info**

Devuelve información general variada sobre el nodo y la red, como la altura, la cantidad de transacciones totales, o el tipo de red.

No hay parámetros de entrada.

En la Fig. 2.3 se muestra la salida de la llamada a este endpoint en la mainnet, ya que la información es más completa que la que tenemos en la testnet.

### **get\_block**

Devuelve toda la información del bloque pedido, ya sea por hash o por altura.

Parámetros de entrada (a elegir entre uno de los dos):

- height
- hash

### **get\_fee\_estimate**

Devuelve una estimación de la comisión por byte actual. Esta es la comisión que el remitente de una transacción debe pagar al minero. Esta comisión es mayor cuanto mayor sea el tamaño en bytes de la transacción.

### **get\_transactions**

Devuelve información sobre las transacciones pedidas por sus hashes.

Parámetros de entrada:

- txs\_hashes: Array de hashes de las transacciones a consultar.
- decode\_as\_json: Variable booleana con la que especificar si se desea obtener información adicional en formato JSON, como por ejemplo los outputs contenidos en los anillos. Siempre la seleccionaremos a *true* ya que esa información es imprescindible para el estudio que realizaremos.

```
{
  "id": "0",
  "jsonrpc": "2.0",
  "result": {
    "alt_blocks_count": 0,
    "block_size_limit": 600000,
    "block_size_median": 300000,
    "block_weight_limit": 600000,
    "block_weight_median": 300000,
    "bootstrap_daemon_address": "",
    "credits": 0,
    "cumulative_difficulty": 40746069032056921,
    "cumulative_difficulty_top64": 0,
    "database_size": 118111600640,
    "difficulty": 143219049520,
    "difficulty_top64": 0,
    "free_space": 18446744073709551615,
    "grey_peerlist_size": 0,
    "height": 2024857,
    "height_without_bootstrap": 0,
    "incoming_connections_count": 0,
    "mainnet": true,
    "nettype": "mainnet",
    "offline": false,
    "outgoing_connections_count": 0,
    "rpc_connections_count": 0,
    "stagenet": false,
    "start_time": 0,
    "status": "OK",
    "target": 120,
    "target_height": 2024330,
    "testnet": false,
    "top_block_hash": "b26dacc6c2f3105c1923b0bcb8a10ae7d6f8f4096a3b628eb629d8ccd28759bd",
    "top_hash": "",
    "tx_count": 6283721,
    "tx_pool_size": 9,
    "untrusted": false,
    "update_available": false,
    "version": "",
    "was_bootstrap_ever_used": false,
    "white_peerlist_size": 0,
    "wide_cumulative_difficulty": "0x90c2557aa63859",
    "wide_difficulty": "0x2158854430"
  }
}
```

**Fig. 2.3** Salida del endpoint get\_info

## get\_outs

Devuelve key y hash de la transacción de los outputs pedidos. El parámetro principal será el index, que es el número por orden que ocupa ese output en la blockchain. Para outputs anteriores a RingCT, el index es independiente para cada cantidad. Por lo tanto, dependiendo de si se está utilizando RingCT, los parámetros de entrada varían.

Parámetros de entrada:

- outputs: Array de objetos output:
  - index (necesario en todos los casos)
  - amount (si se especifica cantidad, se obtiene el output de esa cantidad en el índice indicado, en caso contrario quiere decir que se desea obtener el output a partir del índice global, el cual pertenecería a una transacción de versión 2 utilizando RingCT)

Un ejemplo de la información importante de este método se muestra en la Fig. 2.4.

```
{
  ...
  "outs": [
    {
      "height": 1221,
      "key": "930c672cdabd3650353a219567f0c204c413906d55f5d17627ffc465fcb0cdfc",
      ...
    }
  ],
  ...
}
```

**Fig. 2.4** Obtención de outputs con el endpoint get\_outs

En este caso, al no haber especificado la cantidad, hemos obtenido el primer output que se creó en la main blockchain desde que se introdujo RingCT en el bloque 1220516. En cambio, si especificamos también una cantidad, obtendremos un output previo a la introducción de RingCT (a no ser que pertenezca a una transacción coinbase, las cuales no utilizan nunca RingCT)

## is\_key\_image\_spent

Comprueba si un hash pertenece a una KI previamente gastada. Simplemente consulta si esa KI existe en algún anillo.

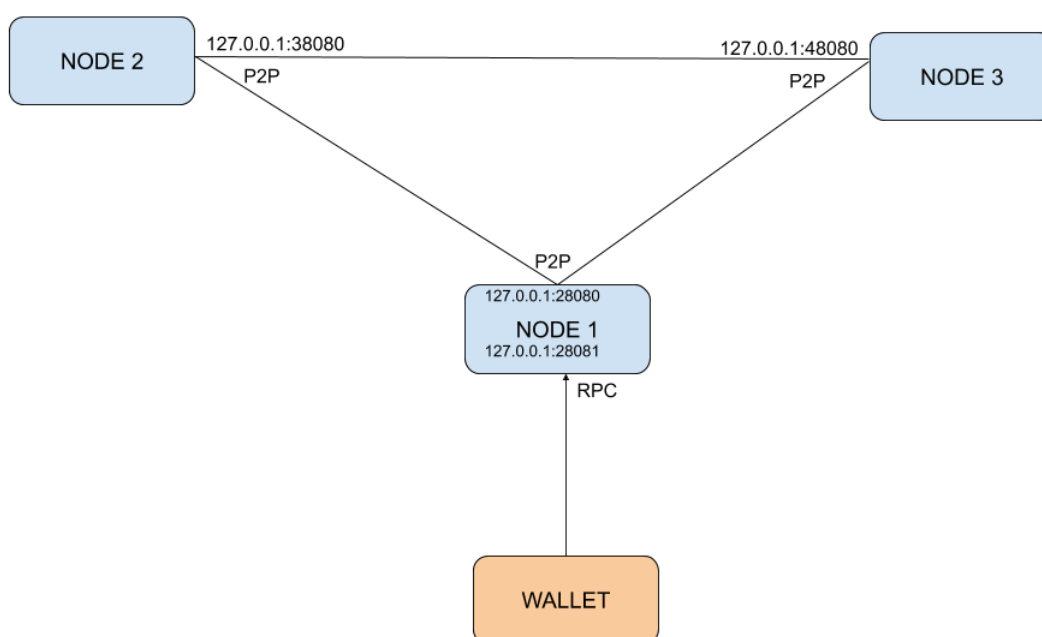
Parámetros de entrada:

- key\_images: Array de hashes a consultar.

Para cada KI devuelve un `spent_status` con valor 1 o 0 según si esa KI se ha encontrado en un anillo o no, respectivamente.

## 2.2. Monero Wallet Command Line Interpreter

Por otro lado, el segundo de los binarios compilados que nos interesan es la implementación de la wallet. El escenario con una wallet se representa en la Fig. 2.5.



**Fig. 2.5** Escenario con wallet conectada a nodo 1

Ya que en para nuestro escenario necesitaremos diferentes wallet y abrirlas y cerrarlas frecuentemente, hemos creado unos scripts para trabajar con ellas de manera rápida y cómoda. Se explican en el anexo C.

Los parámetros que pasamos al archivo `monero-wallet-cli` son los siguientes:

- testnet:** Siempre que el demonio al que nos vayamos a conectar utilice uno de los dos tipos de redes que no sean la principal, debe especificarse tanto en el demonio como en la wallet.
- daemon-address:** Dominio del demonio al que vamos a conectarnos.
- wallet-file:** El archivo que contiene las claves de la wallet.
- trusted-daemon:** Indica que confiamos en el demonio al que nos estamos conectando. En localhost es la opción por defecto, pero en caso contrario el

valor por defecto es `--untrusted-daemon`. El hecho de confiar en el nodo habilita algunas funcionalidades que no estarían disponibles en caso de no confiar, como por ejemplo el minado.

**--password:** La contraseña para abrir la wallet, que en todos los casos hemos dejado vacía. Lo hacemos porque estamos trabajando en una testnet, una wallet de mainnet nunca debería dejarse sin contraseña.

**--log-file:** El directorio en el que se guardarán los logs generados por la wallet. En principio no los necesitaremos, pero siempre es útil disponer de logs.

En la Fig. 2.6 podemos ver la wallet de Alice conectada al nodo local y sincronizada con él. Dispone de más de 10000 moneroj ya que en el momento de la captura ya se había minado con esta wallet.

```

samuel@samuel-msi:~/testnet$ python open_wallet.py alice
This is the command line monero wallet. It needs to connect to a monero
daemon to work correctly.
WARNING: Do not reuse your Monero keys on another fork, UNLESS this fork has key reuse mitigations built in. Doing
so will harm your privacy.

Monero 'Boron Butterfly' (v0.14.1.2-b60cf6a93)
Logging to /home/samuel/testnet/wallets//alice.log
Opened wallet: 9y5Su68RLzkF295NjLyvAU7Ed4DpvxKUtR7tG91XiuzZgtgF49KBi8YGgsePvjLhgbb8bUPiWSGaoSXWRRmpc4Kb4JP9VoJ
*****
Use the "help" command to see the list of available commands.
Use "help <command>" to see a command's documentation.
*****
Background mining not enabled. Run "set setup-background-mining 1" to change.
Starting refresh...
Refresh done, blocks received: 0
Untagged accounts:

```

	Account	Balance	Unlocked balance	Label
*	0 9y5Su6	10746.671914026032	10746.671914026032	Primary account
Total		10746.671914026032	10746.671914026032	

```

Currently selected account: [0] Primary account
Tag: (No tag assigned)
Balance: 10746.671914026032, unlocked balance: 10746.671914026032
Background refresh thread started
[wallet 9y5Su6]:

```

**Fig. 2.6** Wallet de Alice conectada al daemon local

Veamos ahora algunos de los comandos más útiles que utilizaremos en la wallet [18].

**wallet\_info:** Muestra información general sobre la wallet, como el directorio en el que están almacenadas las claves, la dirección, su tipo y el tipo de red.

**status:** Muestra la altura actual de la blockchain y si la wallet está correctamente sincronizada y actualizada.

**fee:** Muestra la comisión actual por byte.

**spendkey:** Muestra el par de claves de gasto.

**viewkey:** Muestra el par de claves de visualización.

**transfer:** La wallet realiza una transferencia a la dirección indicada, especificando la cantidad de moneroj y el tamaño del anillo.

**start\_mining:** La wallet empieza a minar bloques en el demonio.

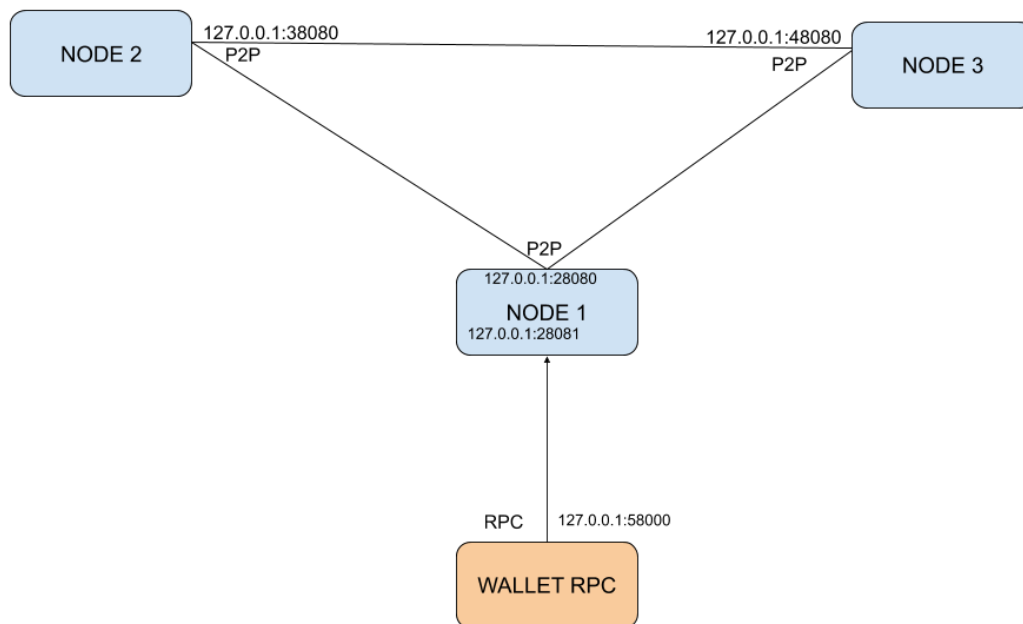
**stop\_mining:** La wallet detiene el minado en el demonio.

**show\_transfers:** Muestra todas las transacciones asociadas a la wallet; muestra todas o bien las filtradas con uno de los siguientes filtros: in | out | pending | failed | pool | coinbase.

**address:** Crea una subdirección.

## 2.3. Monero Wallet RPC

Del mismo modo que el demonio posee una interfaz RPC, la wallet también la posee. En la Fig. 2.7 se muestra el escenario con una wallet RPC conectada al daemon RPC en uno de los nodos.



**Fig. 2.7** Wallet RPC conectada al daemon local

De la misma manera que el demonio, todos los métodos son un HTTP POST contra el contexto /json\_rpc y un posible parámetro llamado "params" en el que incluir los parámetros concretos requeridos por el método en cuestión [19].

## **transfer**

Envía moneroj a una dirección.

Parámetros de entrada:

- amount
- ring\_size
- address
- get\_tx\_key

## **get\_balance**

Consulta el balance de moneroj actual de la wallet en unidades atómicas.

Como parámetros de entrada puede especificarse el índice de la wallet a consultar, pero como no trabajamos con subdirecciones lo dejaremos en el valor por defecto.

## **incoming\_transfers**

Devuelve una lista de las transacciones cuyo destinatario es nuestra wallet.

Parámetros de entrada:

- transfer\_type: Toma los valores "all", "available" o "unavailable", según si se quieren obtener todas las transferencias, sólo aquellas que no se han gastado o solo aquellas que se han gastado, respectivamente. Siempre lo utilizaremos a "all".
- verbose: Valor booleano, si se incluye se obtiene la KI. Siempre lo utilizaremos a true.

## **rescan\_blockchain**

Vuelve a escanear la blockchain y actualiza la información que la wallet tiene guardada localmente. Concretamente, utilizaremos este método para mantener a la wallet informada de qué outputs están gastados.

No hay parámetros de entrada ni de salida. En caso de no recibir un código de error, la operación se ha realizado correctamente.



## 2.4. Adaptación de hard forks en testnet

Necesitaremos modificar el código fuente de Monero para crear nuestros propios hard forks en la testnet. Un hard fork es una ramificación del código fuente [20]. Para ello, hacemos un fork del repositorio oficial de Monero en nuestra cuenta de GitHub y creamos una nueva rama llamada *priv-testnet-ringct*, en la URL siguiente:

<https://github.com/Nessemut/monero/tree/priv-testnet-ringct>

En el archivo *src/hardforks/hardforks.cpp* se declaran tres arrays de struct *hardfork*, uno para cada tipo de red. Estos arrays sirven como referencia al nodo sobre el cual corre el código para saber, en función de la altura actual de la blockchain, qué reglas de versión debe aplicar. El struct consta de cuatro parámetros: *version*, *height*, *threshold* y *time*.

Como nosotros hemos creado una red privada y las alturas no coinciden con las de la testnet oficial, deberemos adaptarlas modificando el array de la testnet. Solo nos interesa que a partir de una altura se utilice RingCT. Pondremos esta altura a 400, ya que permite minar suficientes bloques anteriormente en los que crearemos transacciones sin RingCT. En el argumento versión pondremos la 7, ya que es la versión a partir de la cual se utiliza RingCT. En *time* ponemos el timestamp actual, y el umbral lo dejamos a 0.

En la Fig. 2.8 se muestra la diferencia entre los archivos modificados.

```

sanuel@sanuel-nsi:~/Developer/monero/src/hardforks$ git diff master..priv-testnet-ringct hardforks.cpp
diff --git a/src/hardforks/hardforks.cpp b/src/hardforks/hardforks.cpp
old mode 100644
new mode 100755
index 7ad09dbe..92e62666
--- a/src/hardforks/hardforks.cpp
+++ b/src/hardforks/hardforks.cpp
@@ -64,9 +64,6 @@ const hardfork_t mainnet_hard_forks[] = {
    // version 11 starts from block 1788720, which is on or around the 10th of March, 2019. Fork time finalised on 2019-02-15.
    { 11, 1788720, 0, 1550225678 },

    // version 12 starts from block 1978433, which is on or around the 30th of November, 2019. Fork time finalised on 2019-10-10.
    { 12, 1978433, 0, 1571419200 },
};
const size_t num_mainnet_hard_forks = sizeof(mainnet_hard_forks) / sizeof(mainnet_hard_forks[0]);
const uint64_t mainnet_hard_fork_version_1_till = 1009826;
@@ -75,21 +72,8 @@ const hardfork_t testnet_hard_forks[] = {
    // version 1 from the start of the blockchain
    { 1, 1, 0, 1341378000 },

    // version 2 starts from block 624634, which is on or around the 23rd of November, 2015. Fork time finalised on 2015-11-20. No fork voting occurs for the v2 fork.
    { 2, 624634, 0, 1445355000 },

    // versions 3-5 were passed in rapid succession from September 18th, 2016
    { 3, 800500, 0, 1472415034 },
    { 4, 808219, 0, 1472415035 },
    { 5, 802600, 0, 1472415036 + 86400*180 }, // add 5 months on testnet to shut the update warning up since there's a large gap to v6

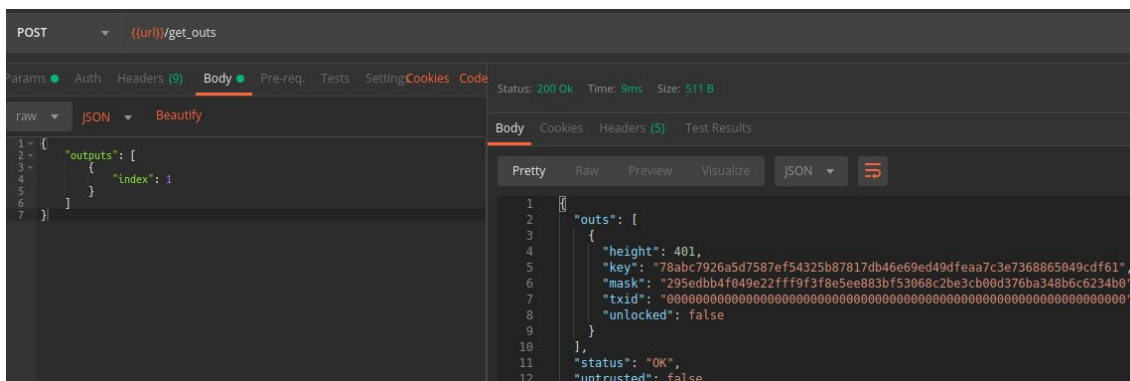
    { 6, 971400, 0, 1501709789 },
    { 7, 1057027, 0, 1512211236 },
    { 8, 1057058, 0, 1533211200 },
    { 9, 1057778, 0, 1533297000 },
    { 10, 1154910, 0, 1550153094 },
    { 11, 1155030, 0, 1550225678 },
    { 12, 1308737, 0, 1569582000 },
+ // setting this to current timestamp and mainnet version makes a private testnet use RingCT from indicated height
+ { 7, 400, 0, 1580946707 },
};
const size_t num_testnet_hard_forks = sizeof(testnet_hard_forks) / sizeof(testnet_hard_forks[0]);
const uint64_t testnet_hard_fork_version_1_till = 624633;
@@ -109,6 +93,5 @@ const hardfork_t stagenet_hard_forks[] = {
    { 9, 177176, 0, 1537821771 },
    { 10, 269000, 0, 1550153694 },
    { 11, 269720, 0, 1550225678 },
    { 12, 454721, 0, 1571419200 },
};
const size_t num_stagenet_hard_forks = sizeof(stagenet_hard_forks) / sizeof(stagenet_hard_forks[0]);

```

Fig. 2.8 Modificación para utilizar RingCT a partir de la altura deseada

Compilamos de nuevo el código. Con los nuevos binarios generados, podremos usar RingCT a partir de la altura 400.

Llamando al método `get_outs` y pidiendo el primer output sin amount, obtendremos el bloque siguiente al hardfork en el que se ha introducido RingCT tal y como ocurre en la main blockchain, en este caso 401.



**Fig. 2.9** Obtención del primer output con amount ofuscado

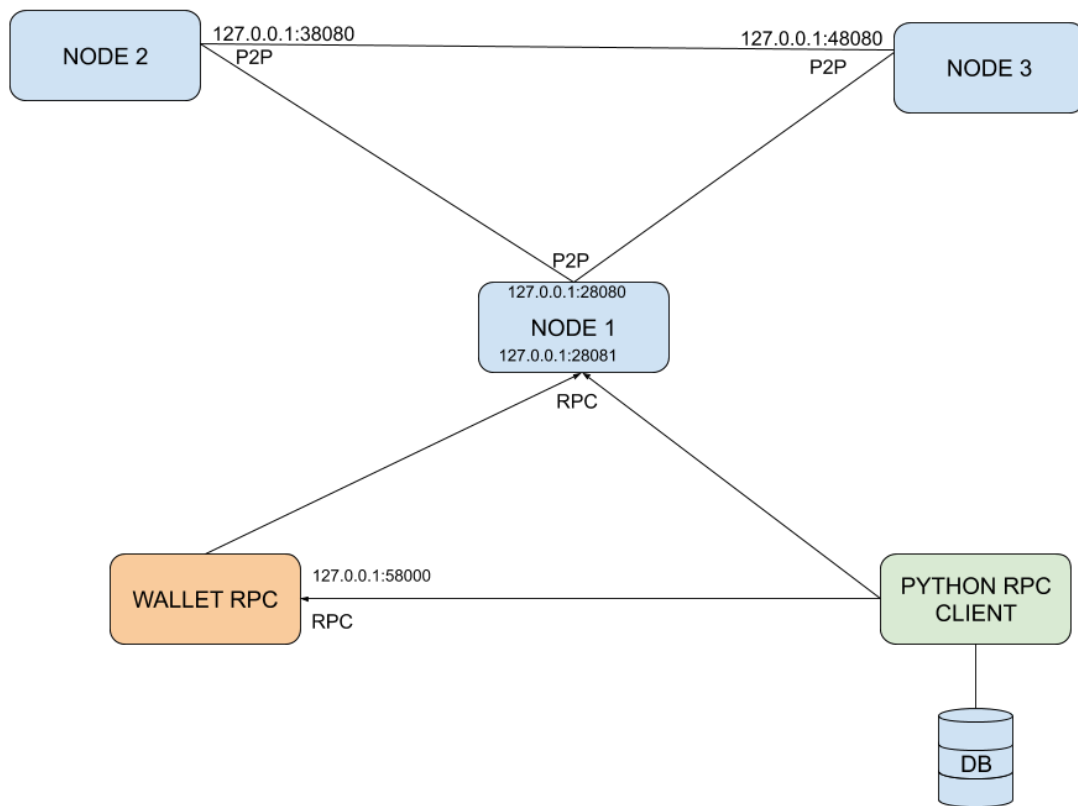
Con esto ya tenemos nuestra propia test blockchain, en la que podremos minar bloques, realizar transacciones entre wallets, consultar transacciones, balances, outputs y cualquier otra operación que podamos necesitar. No obstante, hasta ahora todas estas operaciones las hemos hecho manualmente, ya sea por línea de comandos o consumiendo las APIs. Hemos "jugado" hasta crear una blockchain de altura 2000, la hemos borrado y hemos vuelto a comenzar realizando todo tipo de operaciones durante el proceso, para familiarizarnos con el entorno y las APIs. Llegados a este punto, estamos preparados para automatizar todo este proceso.

## 2.5. Desarrollo de un cliente RPC

Ya tenemos nuestro escenario en funcionamiento, podemos generar nuestras propias blockchains y vamos a proceder a implementar un ataque de inyección de outputs.

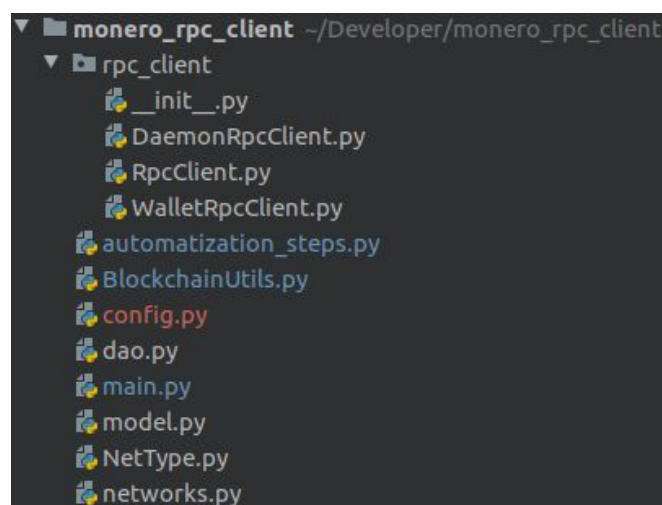
Comenzaremos desarrollando un cliente que consumirá las APIs RPC del nodo y de la wallet. Será desarrollado en Python 3.7 y el código fuente lo alojaremos en control de versiones Git, en [este repositorio de GitHub](#).

El escenario deseado con el cliente corriendo será el mostrado en la Fig. 2.10.



**Fig. 2.10** Escenario completo con APIs RPC y cliente

La estructura de la aplicación se muestra en la Fig. 2.11.



**Fig. 2.11** Estructura de la aplicación desarrollada

### 2.5.1. Consumición de las APIs

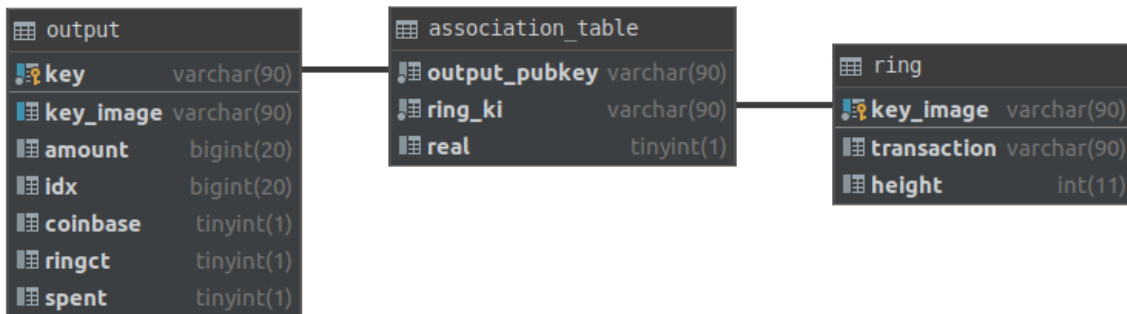
Se han creado las clases `WalletRpcClient` y `DaemonRpcClient` que implementan llamadas a los métodos explicados de la wallet RPC y del daemon RPC en sus respectivos apartados en el capítulo anterior, respectivamente. También se ha creado la superclase `RpcClient` que implementa la lógica común para ambas.

### 2.5.2. Capa de persistencia

A continuación crearemos una base de datos MySQL en la que guardaremos todos los outputs conocidos y los anillos en los que han sido utilizados.

Las dos entidades modeladas en la base de datos serán `Output` y `Ring`, declaradas en el archivo `model.py`. Cada anillo está compuesto por varios outputs y cada output puede aparecer en varios anillos, por lo tanto la relación entre ellas es N:M y se necesitará una tabla de relación. En esta tabla también habrá un valor booleano "real" que especificará si ese output es el real en ese anillo, con valor *null* si se desconoce.

En la Fig. 2.12 se muestra el diagrama UML de esta base de datos.



**Fig. 2.12** Diagrama UML de la base de datos

Para acceder a la base de datos utilizaremos la librería `sqlalchemy` [21], una de las más populares en Python. Haremos uso de ella en la clase `Dao`, en la que tendremos toda la lógica encargada de guardar, leer y modificar los datos necesarios en la base de datos.

### 2.5.3. Capa de servicio

Hemos creado una clase BlockchainUtils, en la que se implementan algunas de las funcionalidades que necesitaremos como capa de servicio. Estos métodos son los siguientes:

**get\_height():** Devuelve la altura actual de la blockchain.

**get\_tx\_count():** Devuelve la cantidad total de transacciones no coinbase en la blockchain.

**get\_blockchain\_array(first, last):** Devuelve una lista de bloques de la blockchain, desde la altura *first* hasta la altura *last*.

**get\_indexes\_from\_offsets\_array():** Cuando se le pide una transacción al demonio, los outputs de sus anillos no son devueltos como su índice absoluto, sino que el primero de ellos lo devuelve en índice absoluto y el resto como un *offset* del anterior. Por ejemplo, si los outputs fuesen [5, 8, 10, 14, 20] la lista recibida sería [5, 3, 2, 4, 6]. Este método convierte de este formato recibido con *offsets* al formato de índices absolutos.

**execute\_once\_a\_block(function):** Ejecuta la función pasada como argumento cada vez que se añade un nuevo bloque a la cadena.

**persist\_coinbase\_transactions(blockchain):** Para cada transacción coinbase en la blockchain que se le pasa como argumento, intenta cargar desde el dao sus outputs. En caso de que no exista, se crea con los atributos necesarios y se persiste desde la funcionalidad aportada por el dao.

**persist\_incoming\_transfers():** Carga todas las transferencias entrantes desde la wallet y persiste sus outputs en la base de datos.

**persist\_rings(blockchain):** Construye una lista de objetos Ring a partir de las transacciones contenidas en los bloques pasados como parámetro. Posteriormente se guardan estos anillos en la base de datos.

**get\_output\_array\_from\_indices\_array(indices):** A partir de una lista de índices de outputs, devuelve otra lista de sus respectivos objetos Output.

**send\_one\_piconero\_to\_myself():** Envía una transacción de 1 piconero a la dirección de la propia wallet.

**save\_output\_array(arr):** Hace uso del dao para guardar en la base de datos todos los outputs en la lista pasada como parámetro.

**plot\_real\_output\_index\_distribution():** Grafica en una imagen la distribución de la antigüedad de los outputs en anillos conocidos.

En este punto ya tenemos un programa completo que nos permite de manera rápida acceder a cualquier dato necesario de la blockchain, tratarlos y mantenerlos en una base de datos, por lo que podemos pasar a analizar la trazabilidad de los outputs tras la implantación de RingCT.

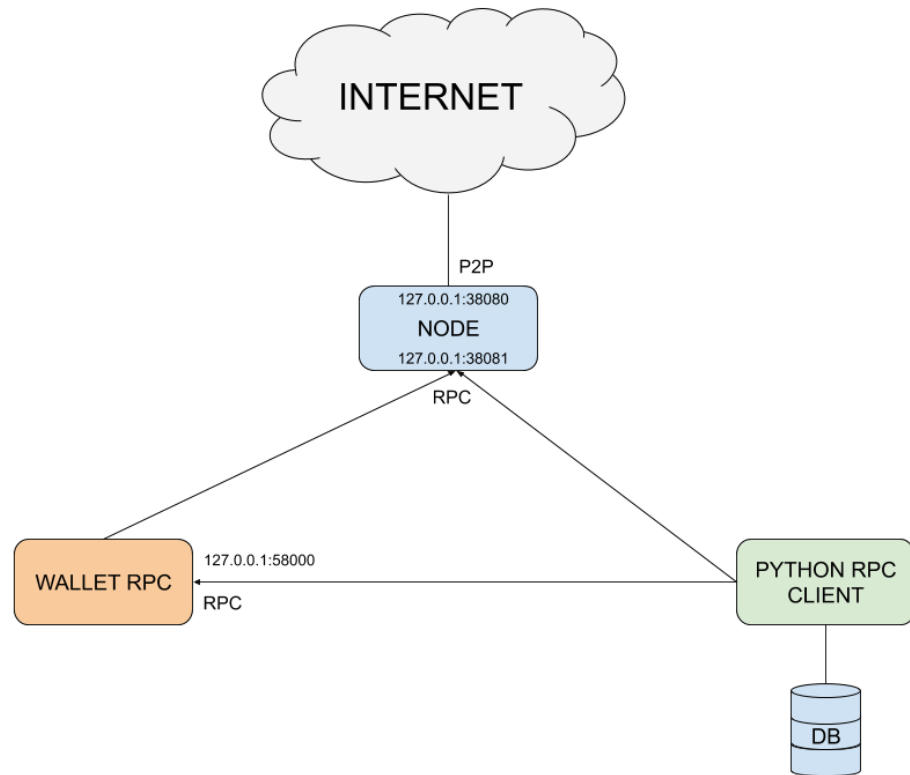
## **CAPÍTULO 3. INTEGRACIÓN DEL CLIENTE RPC EN LA STAGE BLOCKCHAIN Y OBTENCIÓN DE DATOS**

En este capítulo se describe cómo se ha integrado el cliente RPC presentado en el capítulo anterior en la stage blockchain, y cómo se ha generado una base de datos propias de outputs y anillos con valores conocidos (inyección de outputs). El objetivo de dicha inyección es poder determinar con una alta probabilidad, cuál es el output real en un anillo no propio, y por tanto disminuir el nivel de anonimato de Monero. La base de datos obtenida se usará posteriormente como entrada a un algoritmo de Machine Learning en el capítulo 4.

En [15] los autores afirmaban que tras la introducción de RingCT, el nivel de anonimato proporcionado por la firma en anillo aumentaría considerablemente. No obstante este paper data de 2018, momento en el cual RingCT era muy reciente y no realizaron ningún estudio sobre el que apoyar su afirmación. Con estos datos estudiaremos si el anonimato proporcionado por la firma en anillo realmente ha aumentado tras la introducción de RingCT.

### **3.1. Escenario**

Lo primero que debemos hacer es descargar localmente la main y stage blockchains. Este paso debe hacerse con tiempo, puesto que la sincronización y verificación de la blockchain es un proceso que puede durar semanas. Cuando el demonio empieza a descargar la blockchain, cada 100 bloques verifica los hashes para comprobar que no hay ninguna alteración en la blockchain. Por esto, al comienzo es un proceso muy rápido, pero según se van acumulando más bloques, la verificación de cada centena es más lenta ya que hay más bloques anteriores que verificar. En nuestro caso, el primer 30% de la main blockchain se descargó y verificó en menos de dos horas, pero para el último 10% se necesitaron dos semanas. En cambio, la stage blockchain se descargó en pocas horas, puesto que su altura es aproximadamente 5 veces menor.



**Fig. 3.1** Escenario completo en stagenet

Comenzaremos por la stage blockchain por varios motivos. En primer lugar, porque la dificultad de minado de un bloque es suficientemente baja como para que podamos conseguir moneroj de esta manera. De media, podemos minar dos bloques al día, lo cual representa unos 34 moneroj, con lo que tenemos más que suficiente. En cambio, en la main blockchain ni nos planteamos intentar minar, pues podríamos tardar años en minar un bloque. La mejor manera de conseguir moneroj en ella es comprarlos.

## 3.2. Reducción del anonimato usando outputs propios

Los pasos para persistir outputs y posteriormente tratarlos los veremos a continuación. Cada uno de los pasos se corresponde con una de las funciones declaradas en el archivo [monero-rpc-client/automatization\\_steps.py](#)

### 3.2.1. Inyectar outputs

En este primer paso lanzaremos miles de transacciones contra nuestra propia dirección. El objetivo será que estos outputs sean posteriormente utilizados en transacciones de otros usuarios. Para ello se ha implementado un método

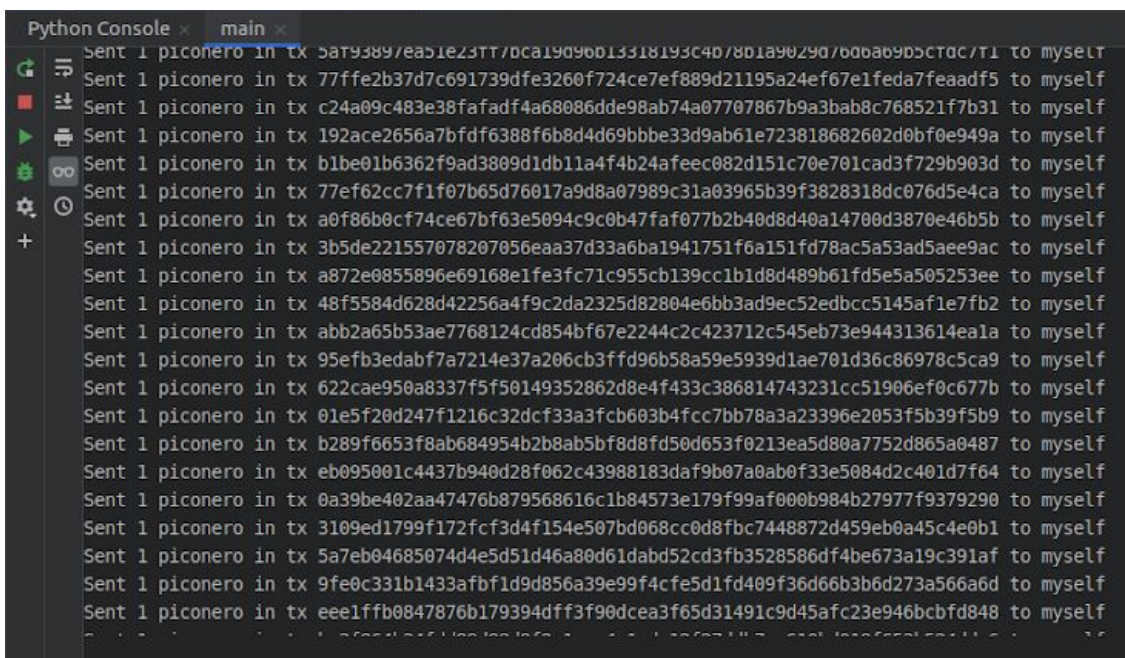


*inject(n)* donde *n* es el número deseado de transacciones de 1 piconero a realizar. Se muestra en la Fig 3.2.

```
def inject(n):
    for i in range(0, n):
        bcutil.send_one_piconero_to_myself()
        if i % 25 == 0:
            wallet.rescan_blockchain()
```

**Fig. 3.2** Método *inject(n)*

Cada 25 envíos de transacciones se realiza un escaneado de la blockchain, ya que la wallet no mantiene un registro de los outputs ya gastados, este registro debe obtenerlo del nodo. Si no se pide periódicamente este registro, pasados una serie de bloques la wallet intentará utilizar outputs ya gastados en sus transacciones y el nodo las rechazará.



The screenshot shows a Python Console window with a list of 25 transactions. Each line starts with 'Sent 1 piconero in tx' followed by a long hexadecimal transaction ID and ends with 'to myself'. The transactions are listed sequentially, demonstrating a mass sending operation.

**Fig. 3.3** Envío masivo de transacciones de 1 piconero

### 3.2.2. Guardar outputs

En este segundo paso guardaremos en la base de datos todos los outputs de los que dispongamos alguna información. Se ha implementado el método *persist\_outputs(start\_block)*, mostrado en la Fig. 3.4.

```
def persist_outputs(self, start_block):
    height = int(self.bcutil.get_height())
    for i in range(start_block, height, INTERVAL):
        blocks = self.bcutil.get_blockchain_array(i, i+INTERVAL-1)
        self.bcutil.persist_coinbase_transactions(blocks)
        collect()

    self.bcutil.persist_incoming_transfers()
```

**Fig. 3.4** Método `persist_outputs(start_block)`

Estos outputs los obtendremos de dos maneras diferentes:

**Guardar transacciones coinbase:** Guardaremos en la base de datos todos los outputs generados en las transacciones que han sido recompensadas a los mineros tras minar un bloque. Todos estos outputs son públicos, no utilizan RingCT y nos los proporciona el demonio.

**Guardar transacciones entrantes:** Persistiremos en la base de datos todos los outputs cuyo destino sea nuestra propia wallet. Estos outputs nos los proporciona la API de la wallet.

Aquellos outputs generados en las transacciones que hemos minado con nuestra wallet serán obtenidos desde ambos procesos.

### 3.2.3. Guardar anillos existentes

A continuación guardamos en la base de datos los anillos conocidos. Será con estos anillos con los que más adelante calcularemos la probabilidad de deducir el output real. Se ha implementado el método `persist_rings()`, mostrado en la Fig. 3.5.

```
def persist_rings(self):
    logging.info('Persisting rings from height {} to {}'.format(self.last_persisted_height, self.working_height))
    for i in range(self.last_persisted_height, self.working_height, INTERVAL):
        blocks = self.bcutil.get_blockchain_array(i, i+INTERVAL-1)
        self.bcutil.persist_rings(blocks)
```

**Fig. 3.5** Método `persist_rings()`

### 3.2.4. Marcar outputs en anillos propios

En este paso marcaremos todos los outputs contenidos en nuestros anillos. Para esto seleccionamos todos los anillos cuya KI coincida con una de nuestras KI, lo que indicará que ese anillo es nuestro ya que el output real es aquel que nosotros conocemos. Marcamos este output como real en la columna 'real' de la tabla 'association\_table' en la base de datos, y todos los demás como falsos. Se ha implementado el método *mark\_my\_rings()*, mostrado en la Fig. 3.6.

```
def mark_my_rings(self):
    logging.info('Marking realness of outputs in own rings')
    my_spent_outputs = self.dao.get_own_spent_outputs()
    for output in my_spent_outputs:
        try:
            ring = self.dao.get_ring(output.key_image)
            for ring_output in ring.outputs:
                self.dao.mark_output_in_ring(
                    ring_output,
                    ring.key_image,
                    ring_output.key_image == output.key_image,
                    True
                )
        except AttributeError:
            # NOTE: this exception might be thrown if the ring is not
            yet persisted
            pass
```

**Fig. 3.6** Método *mark\_my\_rings()*

### 3.2.5. Marcar outputs propios como señuelos

Ahora marcaremos todos nuestros outputs conocidos en otros anillos que no son nuestros, en los que sabemos que han sido seleccionados como señuelos. Se ha implementado el método *mark\_my\_outputs\_in\_other\_rings(start\_block)*, mostrado en la Fig. 3.7.

```
def mark_my_outputs_in_other_rings(self):
    logging.info('Marking own decoy outputs as false')
    my_outputs = self.dao.get_known_outputs()
    for output in my_outputs:
        self.dao.mark_output_in_ring(
            output,
            output.key_image,
            False,
            False
        )
```

**Fig. 3.7** Método `mark_my_outputs_in_other_rings(start_block)`

### 3.2.6. Clasificar anillos

Una vez finalizado el proceso anterior, podemos proceder a buscar si hay algún anillo deducible (es decir, que conocemos 10 outputs como falsos, por lo que el restante será necesariamente el real). En este paso se genera un diccionario de tamaño 11 con claves (1, 2, 3, ..., 11) y como valor la cantidad de anillos cuya cantidad de outputs restantes para ser deducibles es la clave. De esta manera clasificamos los anillos según la cantidad de outputs restantes para ser deducibles. En caso de encontrar anillos deducibles se pasa una lista de sus KI.

Se ha implementado el método `find_output_deducibility(start_block)`, mostrado en la Fig. 3.8.

Como veremos más adelante, encontrar anillos 100% deducibles es extremadamente improbable.

```
def find_output_deducibility(self):
    report = {}
    key_images = self.dao.other_rings_with_at_least_one_own_decoy()
    deducible_outputs = []

    for ki in key_images:
        remaining =
self.dao.get_remaining_outputs_from_ring_with_decoys(ki[0])
        if remaining not in report:
            report[remaining] = 1
        else:
            report[remaining] = report[remaining] + 1
        if remaining == 0:
            deducible_outputs.append(ki)

    report = collections.OrderedDict(sorted(report.items()))
    output_line = 'Reporting found rings with known decoys:'
    for key in report:
        output_line = output_line + '\n\t {} rings with {} outputs
remaining to be deducible'\
        .format(report[key], key)
    logging.info(output_line)
    return deducible_outputs if len(deducible_outputs) > 0 else None
```

**Fig. 3.8** Método find\_output\_deducibility(start\_block)

### 3.2.7. Marcar outputs deducidos en otros anillos

Este último paso se realizará en el caso de que en el paso anterior se haya encontrado algún anillo deducible.

Si hemos encontrado un anillo deducible podemos marcarlo en la base de datos como real. A pesar de que ese output no sea nuestro, por descarte sabemos que es el real. Un output sólo puede ser gastado una vez, por lo tanto, podremos marcar ese output como falso en todos los demás anillos que aparezca. Es la misma operación que se realiza en el capítulo 3.2.5, solo que en esta ocasión los outputs que marcamos no son nuestros.

No obstante, y como se comenta en el apartado siguiente, esta situación es muy infrecuente y debido a la carencia de ejemplos, este método no se ha implementado. Se ha marcado como #TODO en el código como posible futura implementación.

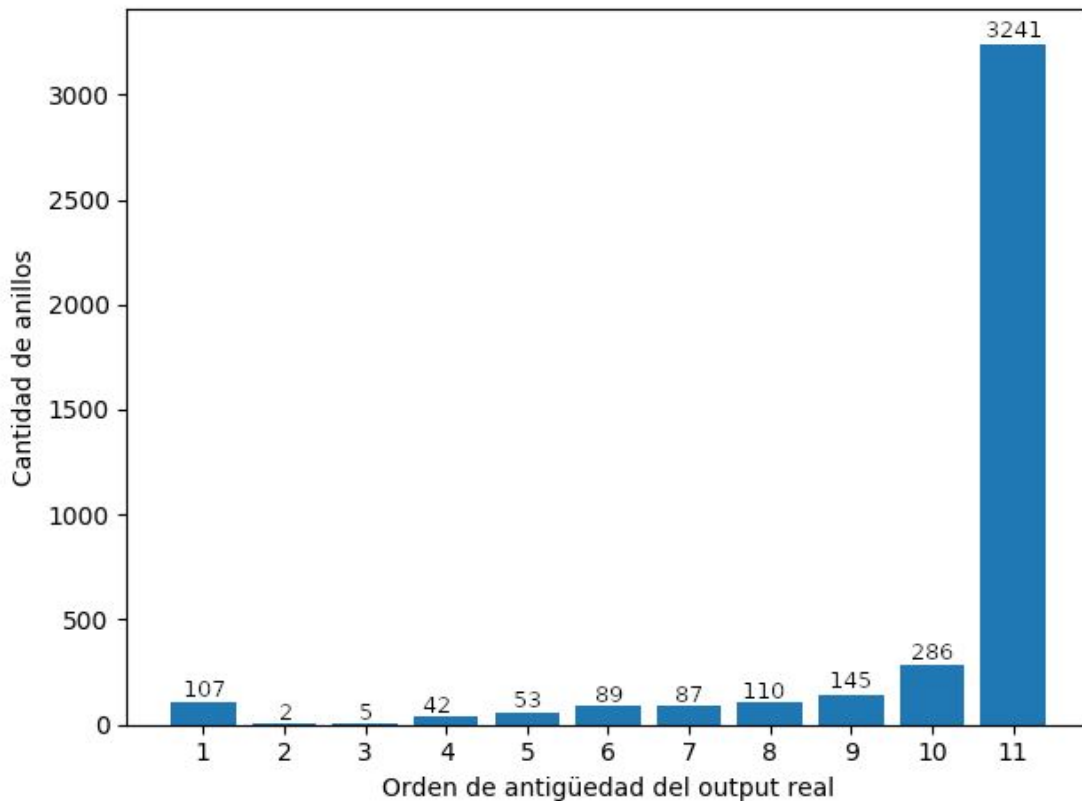
### 3.3. Resultados de la inyección de outputs

Recordemos que en [15] los autores estiman que el porcentaje de anillos cuyo output real es el más reciente es del 80.35%, excluyendo todos los anillos utilizando RingCT. En este análisis, no obstante, hemos analizado únicamente anillos posteriores a RingCT. Los resultados en el momento del análisis, con un total de anillos conocidos de 4167 es que en 3241 de ellos el output real es el más reciente. Porcentualmente es un 77.78%, que es un valor próximo al 80.35% obtenido antes de RingCT.

La tabla siguiente muestra todos los valores para cada orden de antigüedad (siendo 11 el más nuevo y 1 el más antiguo) y en la Fig. 4.3. se grafican para una mejor visualización de los datos.

**Tabla 3.1** Distribución de edad de los outputs reales en anillos

Orden de de edad del output real	Anillos
11	3241
10	286
9	145
8	110
7	87
6	89
5	53
4	42
3	5
2	2
1	107



**Fig. 3.9** Distribución del output real en anillos según el orden de edad

Estos resultados son muy positivos a la hora de intentar deducir el output real de anillos. Sin ninguna otra pista, ya podríamos decir con un 80% de probabilidad de acertar que el output real es el más reciente. No obstante, sí que tenemos más pistas. Hemos estado inyectando miles de outputs, por lo que aunque sea difícil que en un anillo ajeno todos los señuelos sean nuestros, con que hayan algunos ya aumentamos la probabilidad de acierto del output real.

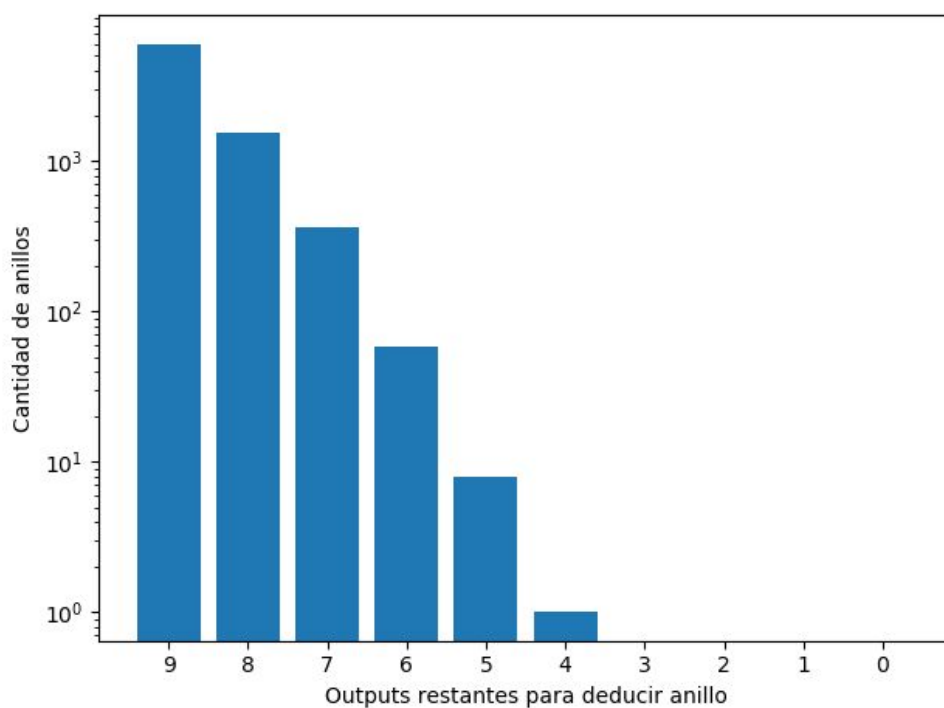
Imaginemos un anillo en el cual se han seleccionado dos de nuestros outputs. Si estos dos outputs fuesen el segundo y tercero más antiguo, según la distribución anterior pasaríamos de tener una probabilidad del 77.78% al 86.75% de que fuera este último.

Si en vez de un anillo con dos, fuese uno con cuatro señuelos nuestros como segunda, tercera, cuarta y quinta posiciones de edad subiríamos la probabilidad de que el output más nuevo sea el más reciente con un 91.58% de probabilidad.

Utilizando el mismo método descrito en el apartado 4.1.6. vamos a observar cuántos anillos ajenos han seleccionado como señuelo algún output propio. Los resultados son los mostrados en la tabla 4.2.

**Tabla 3.2** Cantidad de anillos con uno o más señuelos propios con un total de outputs conocidos de 4167

Outputs restantes para deducibilidad	Anillos
9	5996
8	1545
7	361
6	58
5	8
4	1
3	0
2	0
1	0
0	0



**Fig. 3.10** Outputs restantes para la deducibilidad

Ya hemos visto que en RingCT los outputs reales tampoco están distribuidos de forma homogénea en los anillos. Por lo tanto, podemos proceder a analizar



diferentes maneras de intentar predecir el output real de un anillo. Lo primero que haremos será calcular el porcentaje de anillos en el que se cumple la condición de que el output real sea el más comúnmente repetido como real, descartando outputs que conocemos que son falsos.

### 3.4. Cálculo probabilístico

Por simple estadística y tal como nos sugerían los resultados mostrados en la tabla 4.2, la manera de escoger el output real con mayor probabilidad de acertar sería, de todos los outputs que desconocemos, escoger aquel cuya posición sea la más probable, es decir, la que presenta una mayor frecuencia de aparición. Estas frecuencias de aparición son las mostradas en la Fig. 3.9. En el código mostrado en la Fig 3.11 se ordenan las frecuencias de aparición de outputs reales en una lista y para cada anillo escogemos como real el output desconocido cuya posición aparezca antes en la lista.

```
ordered_probability_positions = [11, 10, 9, 8, 1, 6, 7, 5, 4, 3, 2]

def predict(ring):
    for position in ordered_probability_positions:
        if ring[position-1] == 1:
            return position

def get_empirical_accuracy(x, y):
    test_length = len(x)
    correctly_predicted_count = 0
    for row in range(0, test_length):
        ring = x[row][2:23:2]
        if y[row] == predict(ring):
            correctly_predicted_count += 1

    print(correctly_predicted_count/test_length)
```

**Fig. 3.11** Cálculo probabilístico de deducibilidad

Siguiendo este simple método hemos obtenido una precisión del 86,439%. En el capítulo siguiente utilizaremos algoritmos de ML para intentar mejorar este valor.

## CAPÍTULO 4. ANÁLISIS DE TRAZABILIDAD DE OUTPUTS

Machine Learning [22] es una rama de la inteligencia artificial que permite a una máquina aprender a realizar una cierta tarea sin necesidad de ser programada explícitamente para realizarla. Existen diferentes modalidades de algoritmos, de entre los cuales nosotros utilizaremos el aprendizaje supervisado. En esta modalidad, el algoritmo debe recibir un entrenamiento con muestras conocidas para que aprenda a realizar la tarea deseada. En el aprendizaje no supervisado, en cambio, no es necesario ningún entrenamiento, pues el algoritmo clasifica las muestras por similitud. La tercera modalidad de ML es el aprendizaje reforzado, en el cual se utilizan datos de entrenamiento pero el algoritmo debe ser capaz de añadir más información de la proporcionada a las muestras de entrenamiento. Estos aprenden a base de prueba-error recibiendo feedback continuamente.

Dentro del aprendizaje supervisado encontramos dos tipos: regresión y clasificación. En la regresión el resultado que se espera obtener es un número dentro de un grupo infinito de posibilidades. En la clasificación, en cambio, el resultado es un grupo dentro de un conjunto limitado. Nosotros utilizaremos clasificación, pues lo que nos interesa es clasificar las muestras dentro de uno de los 11 posibles grupos, donde cada grupo será una posición del output real dentro de las 11 posiciones de un anillo.

En los algoritmos que utilizaremos para nuestro análisis haremos uso de tres herramientas para comprobar la efectividad obtenida:

- La *accuracy* o precisión, que indica el porcentaje de muestras clasificadas correctamente sobre el total de muestras totales. Los valores que se presentan son el resultado de hacer la media de 30 iteraciones.
- La matriz de confusión, una matriz cuadrada (en nuestro caso 11x11, pues ese es el tamaño de posibles grupos en los que clasificar las muestras) en la que las columnas representan el número de predicciones en cada grupo y las filas muestran el número de muestras reales en ese grupo. En un caso ideal con precisión del 100% todas las muestras estarían en la diagonal de la matriz.
- El Classification Report, que desglosa por grupos la precisión y la cantidad de muestras clasificadas.

Nosotros utilizaremos la implementación de tres algoritmos implementados en la librería sklearn [23], con el cual seguiremos trabajando con Python 3.7.

En nuestro caso nos interesa que nuestro ordenador aprenda a reconocer el output real de un anillo a partir de cierta información relacionada que le aportemos. Para ello, se usarán dos algoritmos supervisados muy populares:

K-Nearest Neighbors y Random Forest. A pesar de que existen muchos otros algoritmos que podrían aplicarse a este caso en particular, el objetivo de este trabajo era por un lado demostrar que es posible realizar un ataque de inyección de outputs y que dicho ataque permite predecir con una alta probabilidad cuál es el output real en una firma en anillo. La evaluación y optimización de los diferentes algoritmos de ML se deja para futuros trabajos.

Lo primero que necesitaremos para comenzar esta tarea es generar el dataset. Este dataset será un archivo de texto en el que cada línea será una muestra, en nuestro caso la información relevante sobre un anillo y que constituye la entrada del algoritmo a utilizar. El dataset lo dividiremos en dos partes, una parte (entre el 60% y el 80%) serán las muestras con las que entrenaremos, y el resto serán muestras de tests con las que obtener los resultados anteriormente mencionados. Para ello extraeremos los datos que tenemos guardados en la base de datos y con ellos crearemos un archivo .csv con la información que nos interesa.

#### 4.1. Generar datasets

El dataset de entrenamiento se va a generar con nuestros anillos, en los que sabemos cuál es el output real. No obstante, para entrenar nuestro modelo necesitaremos generar un dataset con el mismo formato que el dataset de anillos no conocidos, por lo que a pesar de saber qué output es el real tenemos que etiquetarlo como si no lo supiéramos. Por lo tanto, etiquetaremos como falso todos los outputs que sean nuestros excepto el real. El output real y todo el resto de outputs que no sean nuestros los etiquetaremos como desconocidos. Así que cada output podrá tener dos valores diferentes: 0 (sabemos que es falso) o 1 (desconocemos si es real o falso). Este es el mismo método que se siguió en el capítulo 3.4 para realizar el cálculo probabilístico.

**0:** La altura del bloque que contiene la transacción en la que se encuentra el anillo

**1-22:** 11 pares de valores *altura\_output, valor\_real\_output*.

**23:** Índice de edad del output real.

En la Fig. 4.1 se muestran algunas filas del dataset de entrenamiento generado.

```

577811,477225,1,572802,1,575473,1,576457,1,576658,1,577497,1,577680,1,577692,1,577726,1,577786,1,577801,1,11
543737,34000,1,140962,1,477340,1,537993,1,538012,1,538018,1,538739,1,539324,1,541754,1,542666,1,543725,1,11
523285,34000,1,34000,1,34000,1,34000,1,34000,1,34000,1,476881,1,477013,1,522632,1,522796,1,11
560860,34000,1,34000,1,34000,1,34000,1,434888,1,558076,1,558179,1,560092,1,560332,1,560593,1,560803,1,10
544581,538001,1,538018,1,538024,1,542278,1,542887,1,543291,1,543705,0,543746,1,543790,1,544140,1,544441,1,8
570307,34000,1,34000,1,540545,1,568159,1,570032,1,570095,1,570147,1,570293,1,570296,1,570296,1,570297,1,11
537985,34000,1,34000,1,34000,1,367265,1,454244,1,499168,1,536934,1,537966,1,537971,1,537972,1,8
544693,141191,1,435484,1,435499,1,476836,1,538008,1,538738,1,540609,1,542384,1,542881,1,544661,0,544683,1,11
558768,34000,1,137029,1,548096,1,551855,1,554479,1,555505,1,555713,1,558167,1,558178,1,558216,0,558758,1,11
558708,34000,1,34000,1,303676,1,367232,1,453443,1,554406,1,556632,1,557462,1,558611,0,558615,0,558698,1,11
544585,141011,1,537991,1,537997,1,538739,1,538756,1,539879,1,542418,1,542801,1,542848,1,543747,1,544141,1,10
557435,34000,1,34000,1,34000,1,34000,1,34000,1,548641,1,553338,1,557263,1,557308,1,557315,1,557319,1,8
559755,34000,1,34000,1,34000,1,34000,1,34000,1,476642,1,505627,1,557315,1,559704,0,559745,1,11
571315,367130,1,435001,1,570732,1,570798,1,570826,1,570992,1,571023,1,571292,1,571296,1,571302,1,571302,1,10

```

**Fig. 4.1** Dataset de entrenamiento

Con el dataset generado podemos comenzar el análisis con un algoritmo sencillo, el K-Nearest Neighbors. A partir de este punto, el código utilizado consistirá en un proyecto diferente que está subido en [este repositorio](#).

## 4.2. K-Nearest Neighbors

El algoritmo de K-Nearest Neighbors (KNN) [24] clasifica las muestras en función de las distancias entre la muestra a clasificar y las muestras de entrenamiento más cercanas. Las distancias se pueden calcular de diferentes maneras, nosotros utilizaremos la distancia euclídea por defecto. Probaremos diferentes valores de `n_neighbors`, el cual permite configurar la cantidad de muestras cercanas que se tienen en cuenta para clasificar. Este proceso de probar diferentes valores para encontrar el óptimo se llama *parameter tuning*.

Cargaremos el dataset y realizaremos una prueba para obtener el valor de `n_neighbors` que obtenga mejor precisión. En la Fig. 4.2. se ven los resultados de este primer estudio.

Accuracy: 0.8152046783625732											Classification Report:				
Confusion Matrix:												precision	recall	f1-score	support
[	4	0	0	0	0	0	0	0	0	7]	1	0.40	0.36	0.38	11
[	0	0	0	0	0	0	0	0	0	1]	2	0.00	0.00	0.00	1
[	0	0	0	0	0	0	0	0	0	4]	3	0.00	0.00	0.00	4
[	0	0	0	0	0	0	0	0	0	8]	4	0.00	0.00	0.00	8
[	0	0	0	0	0	0	0	0	0	11]	5	0.00	0.00	0.00	11
[	0	0	0	0	0	0	0	0	0	18]	6	0.00	0.00	0.00	18
[	0	0	0	0	0	0	0	0	0	16]	7	0.00	0.00	0.00	16
[	0	0	0	0	0	0	0	0	0	21]	8	0.00	0.00	0.00	21
[	0	0	0	0	0	0	0	0	2	27]	9	0.00	0.00	0.00	29
[	0	0	0	0	0	0	0	0	9	37]	10	0.82	0.20	0.32	46
[	6	0	0	0	0	0	0	0	0	684]]	11	0.82	0.99	0.90	690
											micro avg	0.82	0.82	0.82	855
											macro avg	0.19	0.14	0.14	855
											weighted avg	0.71	0.82	0.75	855

**Fig. 4.2** Clasificación con KNN

El resultado de este estudio con KNN es insatisfactorio. Se han probado valores entre 3 y 100, y la máxima precisión obtenida ha sido con valores entre 19 y 35. Son esos valores los que se muestran en la Fig. 4.2. Como se puede ver en la imagen el algoritmo no ha aprendido correctamente. Prácticamente se ha limitado a clasificar todas las muestras en el grupo 11 (como se puede ver tanto en la matriz de confusión en que la mayoría de columnas están compuestas por ceros, y en el classification report, en el que la precisión es cero en todas las categorías entre 2 y 9), y como ya sabíamos que aproximadamente un 80% de los anillos tienen su output real en esta última posición, como precisión se obtiene un valor similar.

Con este método hemos obtenido un resultado incluso inferior al resultado probabilístico del apartado anterior. Claramente necesitamos otro modo de intentar clasificar las muestras, por este motivo se ha decidido probar con otro algoritmo como el Random Forest.

### 4.3. Decision Tree y Random Forest

Un árbol de decisión o Decision Tree (DT) [25] es una herramienta que permite clasificar muestras de un dataset tomando una serie de decisiones utilizando un esquema de árbol. La raíz del árbol sería la muestra a clasificar, los nodos intermedios representarían cada una de las decisiones a tomar y las hojas del árbol serían los grupos en los que la muestra podría clasificarse.

En la Fig. 4.3 se muestra un resultado medio obtenido con DT.

Accuracy: 0.8643790849673203											Classification Report:				
<div>Confusion Matrix:</div> <div><div><div>[ [ 15 1 0 0 0 0 0 0 0 1 0 2]</div><div>[ 0 2 0 0 0 0 0 0 0 0 0 0]</div><div>[ 0 2 0 0 1 0 0 0 0 0 0 2]</div><div>[ 0 1 0 1 0 0 0 0 0 1 0 2]</div><div>[ 0 0 0 1 8 0 2 0 0 0 0 2]</div><div>[ 0 0 0 1 1 8 3 2 0 0 0 2]</div><div>[ 0 0 0 0 1 1 4 3 0 0 0 4]</div><div>[ 0 0 0 0 0 1 7 6 6 3 6]</div><div>[ 0 0 0 0 0 0 3 2 7 5 11]</div><div>[ 4 0 0 0 0 0 0 1 11 23 28]</div><div>[ 0 0 0 0 3 6 0 6 10 17 984]]</div></div></div>											precision	recall	f1-score	support	
											1	0.79	0.79	0.79	19
											2	0.33	1.00	0.50	2
											3	0.00	0.00	0.00	5
											4	0.33	0.20	0.25	5
											5	0.57	0.62	0.59	13
											6	0.50	0.47	0.48	17
											7	0.21	0.31	0.25	13
											8	0.30	0.21	0.24	29
											9	0.19	0.25	0.22	28
											10	0.48	0.34	0.40	67
											11	0.94	0.96	0.95	1026
											micro avg	0.86	0.86	0.86	1224
											macro avg	0.42	0.47	0.43	1224
weighted avg	0.86	0.86	0.86	1224											

**Fig. 4.3** Clasificación con Decision Tree

En las diferentes simulaciones e independientemente del porcentaje de muestras usadas para test y random\_state (la variación de este parámetro asegura una variación en los resultados) hemos comprobado que el valor es siempre muy próximo al 86%, que es el mismo valor que obteníamos con el cálculo probabilístico.

No obstante, hay un algoritmo que utiliza DT que nos puede ser más útil. Este algoritmo es el Random Forest (RF). Un RF es un algoritmo que combina distintos DTs, entrenando cada uno con una porción distinta del dataset y selecciona la mejor selección de entre los diferentes árboles [26].

Se ha intentado optimizar el funcionamiento del algoritmo variando algunos de sus parámetros. En particular, se han hecho 75 pruebas distintas en las que se han probado valores de n\_estimators (cantidad de árboles) entre 5 y 150 en intervalos de 10 y max\_depth (profundidad de cada árbol, es decir el número máximo de niveles de nodos intermedios) entre 0 y 33. Cada punto en la Fig. 4.4 muestra el par de valores con el cual la precisión ha sido mayor en cada una de las 75 pruebas.





Con RF entrenando con un dataset de 4240 muestras, con una proporción de muestras de test del 30% y  $n\_estimators = 95$  y  $max\_depth = 18$  obtenemos precisiones de entre 89% y 94.575%. En el Classification Report podemos ver que en algunas de las categorías más bajas tenemos valores bajos, o incluso 0 en el caso de las categorías 2 y 3. Esto es debido a la baja cantidad de muestras de esas categorías, las cuales no han sido suficientes para que el algoritmo aprenda correctamente.

En la Fig. 4.6 mostramos algunos anillos que han sido correctamente deducidos por el algoritmo entrenado. El formato en que se muestra es el siguiente:

[outputs del anillo por edad], output real verdadero, output real predicho

```
[1 1 1 1 1 1 1 1 1 1] 4 4
[1 1 1 1 1 1 1 1 1 1] 9 9
[1 1 1 1 1 1 1 1 1 1] 8 8
[1 1 1 1 1 1 1 1 1 1] 7 7
[1 1 1 1 1 1 1 1 1 1] 10 10
[1 1 1 1 1 1 1 0 1 1] 9 9
[1 1 1 1 1 1 1 1 1 1] 1 1
[1 1 1 1 1 1 1 1 1 1] 5 5
[1 0 1 1 1 1 1 1 1 1] 7 7
[1 1 1 1 1 1 1 1 0 1] 10 10
[1 1 1 1 1 1 1 1 1 1] 4 4
[1 1 1 1 1 1 1 1 1 1] 6 6
[1 1 1 1 1 1 1 1 1 1] 10 10
[1 1 1 1 1 1 1 1 1 1] 6 6
[1 1 1 1 1 1 0 1 1 1] 1 1
[1 1 1 1 1 1 1 1 0 1] 6 6
[1 1 1 1 1 1 1 1 1 1] 6 6
```

**Fig. 4.6** Anillos predichos con RF

Como se ve en la Fig. 4.6 nuestro RF ha aprendido a escoger los outputs reales de una manera mucho más acertada. Miremos por ejemplo este: [1 1 1 1 1 1 1 1 1 1] 4 4. En este caso tenemos un anillo del cual el algoritmo no tenía ninguna pista sobre la falsedad de ningún output, en el que nosotros sí sabemos que el output real es el cuarto. Siguiendo de nuevo la estadística que nos arroja la tabla 4.2, la probabilidad de que en un anillo del cual no conocemos ningún output el real sea el 4 es del 1% (42/4167). A pesar de esta probabilidad tan baja, nuestro modelo lo ha conseguido predecir correctamente. Esta misma situación ha ocurrido también en otros anillos con el output real en posiciones 1, 5, 6, 7, 8, 9, 10 y 11. Para anillos con en el output real en posición 2 y 3 no teníamos ninguna muestra para entrenar puesto que son muy infrecuentes.



A pesar de seguir teniendo entre un 6% y un 10% de muestras que no se están clasificando correctamente, si hemos conseguido mejorar la proporción de muestras correctas que deduciríamos de manera empírica. En lo que en un principio y de manera teórica debería ser una probabilidad del 9.091% de acertar el output real de un anillo (escogiendo uno de entre 11 al azar) nosotros hemos conseguido multiplicar esta probabilidad por 10, reduciendo considerablemente el anonimato aportado por la firma en anillo.

#### 4.4. Análisis teórico en mainnet

Todo el trabajo llevado a cabo hasta ahora se ha realizado sobre, en primer lugar, la testnet (para el desarrollo de la aplicación) y sobre la stagenet (para generar anillos y extraer los datos). Ahora bien, estas redes son solo un entorno de pruebas para la red real, la mainnet. Es en esta mainnet donde los moneroj tienen valor reconocido por la comunidad. La stagenet, a nivel de código fuente es una copia exacta de la mainnet. No obstante, existen diferencias entre el uso de ambas que podrían alterar el comportamiento del modelo que hemos obtenido, haciendo que no fuese tan efectivo en la mainnet. Principalmente esta diferencia es el hecho de que en la mainnet hay una cantidad de transacciones muy superior, de manera que cuando se forma un anillo y hay que elegir los diez señuelos, estos pueden estar distribuidos de una manera más compensada de lo que hemos visto en la stagenet. Desconocemos si esta diferencia afectaría negativamente al modelo puesto que no sabemos exactamente qué proceso lleva a cabo el random forest para realizar su clasificación, pero es una posibilidad que no debemos descartar. El principal motivo por el que se han realizado el proceso de generación de datos en la stagenet en lugar de la mainnet es debido a que en esta los moneroj son fácilmente minables, en cambio en la mainnet es tan improbable minar un bloque que deberíamos haber comprado moneroj. En otras palabras, el motivo es económico. A continuación vamos a calcular el coste que habría tenido en la mainnet entrenar un modelo tal y como lo hemos hecho.

Primero, utilizando la wallet cli, consultaremos el balance de Monero del que dispone actualmente la wallet utilizada. Simplemente necesitamos ejecutarla ya que este valor se muestra al iniciarla. Esta cantidad es de 696.9176230177112 moneroj.

A continuación consultaremos todas las transacciones coinbase recibidas (es decir, recompensas por bloques minados) utilizando el comando `show_transfers coinbase`. Esto nos da una lista de transacciones que tenemos que procesar para obtener el valor total.

```

Untagged accounts:
  Account      Balance      Unlocked balance      Label
  *    0 5ACfUH  696.300280837711    696.300280837711    Primary account
-----
      Total    696.300280837711    696.300280837711
Currently selected account: [0] Primary account
Tag: (No tag assigned)
Balance: 696.300280837711, unlocked balance: 696.300280837711
Background refresh thread started
[wallet 5ACfUH]: show_transfers coinbase
522796 block unlocked 2020-02-22 18:39:25 13.394434913120 38fc543321878f07a91
fQRNbDcfNZDyGTB1H8mhGVWsG8Na16JH3XRSz7CAXww5519eS48t:13.394434913120 0 -
522812 block unlocked 2020-02-22 19:01:00 13.393868188074 f298c4a7f67a311a8a1
fQRNbDcfNZDyGTB1H8mhGVWsG8Na16JH3XRSz7CAXww5519eS48t:13.393868188074 0 -
522814 block unlocked 2020-02-22 19:02:55 13.393817094571 4bb6317042d6886794a
fQRNbDcfNZDyGTB1H8mhGVWsG8Na16JH3XRSz7CAXww5519eS48t:13.393817094571 0 -
532573 block unlocked 2020-03-07 22:43:28 13.146812799626 a9cd0219e2898e9178f
fQRNbDcfNZDyGTB1H8mhGVWsG8Na16JH3XRSz7CAXww5519eS48t:13.146812799626 0 -
532578 block unlocked 2020-03-07 22:48:13 13.146687422327 a524a1b10a163c82b6e
fQRNbDcfNZDyGTB1H8mhGVWsG8Na16JH3XRSz7CAXww5519eS48t:13.146687422327 0 -
532581 block unlocked 2020-03-07 22:50:48 13.146612196522 8a91a7f4eecef9354d6

```

**Fig. 4.7** Consulta de transacciones coinbase en wallet CLI

Copiamos esa lista en un archivo de texto y con un script sumamos el valor de todas las transacciones.

```

f = open("log.txt", "r")
total = 0
for line in f:
    total += float(line[56:71])
print(total)

```

**Fig. 4.8** Cálculo del total de Monero invertido en stagenet

La cantidad obtenida es de 696.300280837711 moneroj.

En el proceso realizado con esta wallet, todos los moneroj que hemos recibido han sido a través de transacciones coinbase, y todos los moneroj enviados han sido a la misma wallet. Por lo tanto, todos los moneroj perdidos han sido utilizados como comisiones en las transacciones enviadas. La diferencia entre ambas cantidades obtenidas es la cantidad de moneroj pagados como comisión:

$$696.9176230177112 - 696.300280837711 = 0.61734218$$

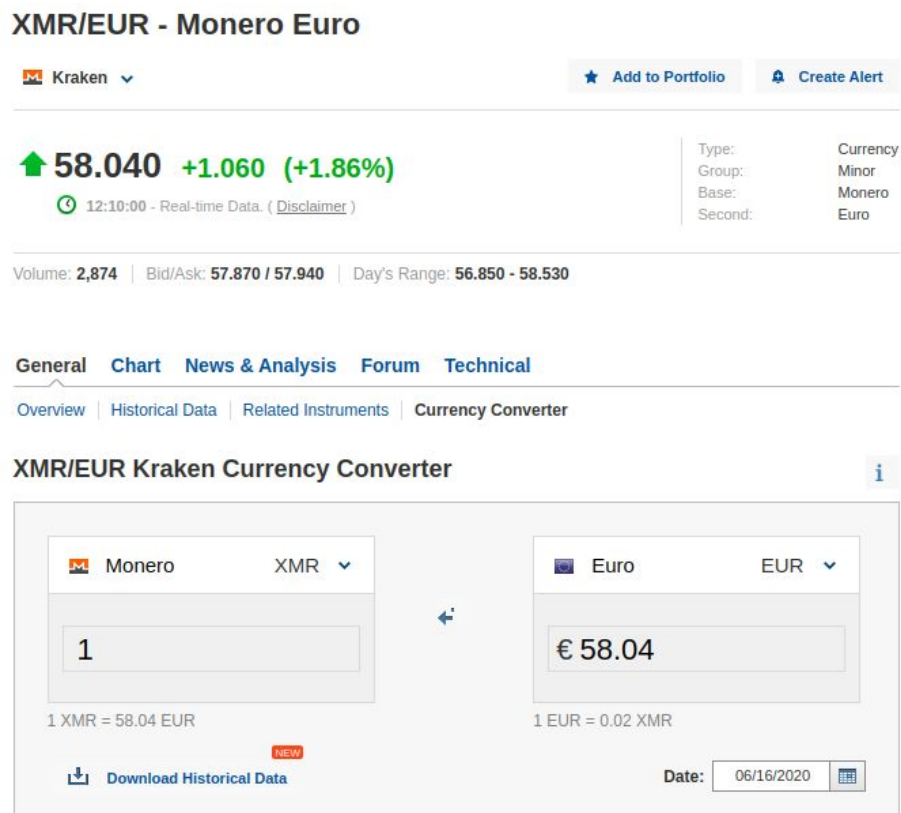
Por otra parte, también hay que tener en cuenta que las comisiones varían entre redes. Para conocer la comisión en cada red utilizamos en cada red el método `get_fee_estimate` explicado en el apartado 2.1.1. Los resultados son que las comisiones en piconero por byte son las siguientes:

Mainnet: 10706 piconero/byte, Stagenet: 83472 piconero/byte

Con una regla de tres obtenemos la comisión que habríamos pagado en la mainnet:

$$0.61734218/83472 * 10706 = 0.07917942997$$

Y finalmente, convertiremos de Monero a Euro consultando los precios de mercado en una web de trading de criptomonedas [27].



**Fig. 4.9** Valor de un monero en euros

En el momento en que se ha consultado, el valor en el mercado de un monero es de 58.04 €.

$$0.07917942997 \text{ monero} * 58.04 \text{ €/monero} = 4.59 \text{ €}$$

Por lo que podemos concluir que haber generado un dataset con unas 4200 muestras habría sido muy barato, 4.59 € en el momento de redacción. En caso

de querer entrenar un modelo en la mainnet, se podría hacer con un desembolso económico insignificante.

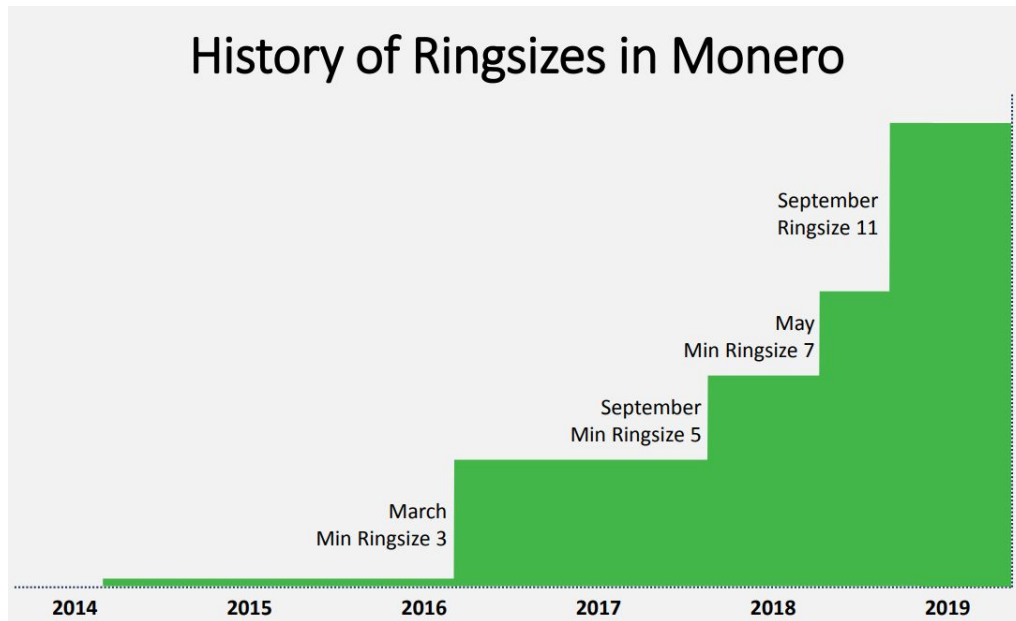
## CAPÍTULO 5. CONCLUSIONES Y DESARROLLO FUTURO

Blockchain es un sistema de base de datos descentralizada, transparente e inmutable que se ha popularizado durante los últimos años. Si bien puede ser utilizado en muchos ámbitos, el más común es utilizarla en transacciones monetarias. La más famosa de estas implementaciones fue Bitcoin. No obstante y a pesar de las ventajas ofrecidas por Bitcoin también presentaba una serie de desventajas, entre las cuales destaca una falta de anonimato. Monero es otra criptomoneda creada unos años más tarde con el objetivo de aportar soluciones a esta falta de anonimato. Para ello utiliza diferentes capas de criptografía, como la firma en anillo o RingCT, una característica que oculta la cantidad de divisa enviada en las transacciones.

En este trabajo hemos creado nuestro propio entorno de trabajo poniendo en marcha una blockchain de test para familiarizarnos con su funcionamiento, y haciendo uso de la misma se ha desarrollado un cliente que se comunica con la blockchain de Monero y realiza en ella las operaciones deseadas. Con este cliente hemos lanzado miles de transacciones y hemos extraído la información de posteriores transacciones que han incluido nuestros outputs en sus firmas en anillo. Este análisis ya se había realizado anteriormente en un estudio [15] anterior a la introducción de RingCT. Nosotros hemos partido de sus resultados para estudiar el nivel de anonimato que realmente puede aportar la característica RingCT.

Los resultados han sido muy similares a los obtenidos en [15] con la red sin RingCT. En ambos casos, en aproximadamente un 80% de los anillos el output real era el más reciente. Utilizando técnicas de ML hemos entrenado diferentes algoritmos utilizando las transacciones conocidas. Con este método hemos podido deducir el output real de hasta el 94% de los anillos analizados.

La conclusión obtenida de estos resultados es que probablemente Monero necesite ampliar el tamaño de sus anillos para poder garantizar el anonimato en las transacciones. El tamaño de los anillos se ha ido incrementando durante los años tal y como mostramos en la Fig. 5.1.



**Fig. 5.1** Historial por años del tamaño de los anillos en Monero [28]

Una posible línea de desarrollo futura sería crear un fork de Monero con un tamaño de anillo superior al actual y repetir los análisis llevados a cabo en este estudio con el objetivo de observar la influencia del tamaño de un anillo en la deducibilidad de este.

Por otra parte, el objetivo principal de este trabajo era encontrar una vulnerabilidad en Monero y analizar la viabilidad de un posible ataque. Machine Learning ha sido una herramienta que se ha utilizado para ese fin, pero sin llegar a profundizar en ello. Con un análisis sencillo como el que hemos realizado ya se han obtenido resultados muy contundentes, por lo que en un trabajo especialmente enfocado en este ataque y haciendo especial énfasis en la utilización de ML es muy posible que los resultados fuesen todavía mejores.

Finalmente, otra posible línea de trabajo futuro sería alojar la aplicación desarrollada en un servidor público de manera que cualquier usuario pudiese lanzar y marcar sus transacciones, creando así una base de datos colectiva con el que llevar a cabo ataques de trazabilidad más agresivos que el que nosotros hemos realizado individualmente y con un hardware limitado.

## BIBLIOGRAFÍA

- [1] Satoshi Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System" (2008) [PDF]. Disponible en: <https://bitcoin.org/bitcoin.pdf>
- [2] Don Tapscott y Alex Tapscott, "La Revolución Blockchain", en *Prólogo*, p. 20, Ediciones Deusto. (2017).
- [3] *Fiat Money*. Disponible en: <https://www.investopedia.com/terms/f/fiatmoney.asp>
- [4] A. Kak, "Public-Key Cryptography and the RSA Algorithm", en *Computer and Network Security* (2020). Disponible en: <https://engineering.purdue.edu/kak/compsec/NewLectures/Lecture12.pdf>
- [5] Kurt M. Alonso, 'KOE', "Zero to Monero" (2018). Disponible en: <https://www.getmonero.org/library/Zero-to-Monero-1-0-0.pdf>
- [6] L. Ismail, H. Materwala, "A Review of Blockchain Architecture and Consensus Protocols: Use Cases, Challenges, and Solutions", en *MDPI* (2019). Disponible en: [https://www.researchgate.net/publication/336057045\\_A\\_Review\\_of\\_Blockchain\\_Architecture\\_and\\_Consensus\\_Protocols\\_Use\\_Cases\\_Challenges\\_and\\_Solutions](https://www.researchgate.net/publication/336057045_A_Review_of_Blockchain_Architecture_and_Consensus_Protocols_Use_Cases_Challenges_and_Solutions)
- [7] Explorador de bloques de Bitcoin. Disponible en: <https://www.blockchain.com/explorer>
- [8] *Hashcash*. Disponible en: <https://en.wikipedia.org/wiki/Hashcash>. Último acceso: Julio 2020
- [9] Alan T. Norman, "Blockchain Technology Explained: The Ultimate Beginner's Guide About Blockchain Wallet, Mining, Bitcoin, Ethereum, Litecoin, Zcash, Monero, Ripple, Dash, IOTA and Smart Contracts" (2017)
- [10] Explorador de bloques de Monero. Disponible en: <https://moneroblocks.info/>
- [11] Nico 'SerHack.', "Mastering Monero", Cap. 5.4.3 en *Ring signature*, LernoLibro LLC. (2018).

- [12] Subaddresses. Disponible en:  
<https://github.com/monero-project/monero/pull/2056>
- [13] Nico 'SerHack.', "Mastering Monero", Cap. 5.4.2 en *Ring Confidential Transactions*, LernoLibro LLC. (2018).
- [14] Panda Adaptive Defense 360, "Cryptojacking: Un coste oculto" (2018). Disponible en:  
[https://www.pandasecurity.com/spain/mediacenter/src/uploads/2018/10/Whitepaper-cryptojacking\\_ES.pdf](https://www.pandasecurity.com/spain/mediacenter/src/uploads/2018/10/Whitepaper-cryptojacking_ES.pdf)
- [15] Möser, M, Soska, K, Heilman, Ethan, L et al. en "An Empirical Analysis of Traceability in the Monero Blockchain" en *Proceedings on Privacy Enhancing Technologies* (2018). Disponible en:  
<https://arxiv.org/pdf/1704.04299/>
- [16] Documentación oficial del daemon de Monero. Disponible en:  
<https://monerodocs.org/interacting/monerod-reference/>
- [17] Documentación oficial del daemon RPC. Disponible en:  
<https://web.getmonero.org/resources/developer-guides/daemon-rpc.html>
- [18] Documentación oficial de la wallet CLI. Disponible en:  
<https://monerodocs.org/interacting/monero-wallet-cli-reference/>. Último acceso: Julio 2020
- [19] Documentación oficial de la wallet RPC. Disponible en:  
<https://web.getmonero.org/resources/developer-guides/wallet-rpc.html>. Último acceso: Julio 2020
- [20] Documentación oficial del repositorio de código de Monero. Disponible en:  
<https://github.com/monero-project/monero/blob/master/README.md>. Último acceso: Julio 2020
- [21] Documentación oficial de SQLAlchemy. Disponible en:  
<https://docs.sqlalchemy.org/en/13/dialects/mysql.html>. Último acceso: Julio 2020
- [22] Shai Shalev-Shwartz y Shai Ben-David, "*Understanding Machine Learning: From Theory to Algorithms*", Cambridge University Press



- (2014). Disponible en:  
<https://www.cse.huji.ac.il/~shais/UnderstandingMachineLearning/>
- [23] Documentación oficial de sklearn. Disponible en:  
<https://scikit-learn.org/stable/index.html>. Último acceso: Julio 2020
- [24] Tutorial de KNN. Disponible en:  
[https://www.tutorialspoint.com/machine\\_learning\\_with\\_python/machine\\_learning\\_with\\_python\\_knn\\_algorithm\\_finding\\_nearest\\_neighbors.htm](https://www.tutorialspoint.com/machine_learning_with_python/machine_learning_with_python_knn_algorithm_finding_nearest_neighbors.htm).  
Último acceso: Julio 2020
- [25] Documentación de Decision Tree. Disponible en:  
<https://scikit-learn.org/stable/modules/tree.html#tree>. Último acceso: Julio 2020
- [26] Tutorial de Random Forest. Disponible en:  
[https://www.tutorialspoint.com/machine\\_learning\\_with\\_python/machine\\_learning\\_with\\_python\\_classification\\_algorithms\\_random\\_forest.htm](https://www.tutorialspoint.com/machine_learning_with_python/machine_learning_with_python_classification_algorithms_random_forest.htm).  
Último acceso: Julio 2020
- [27] Conversor de divisas. Disponible en:  
<https://www.investing.com/crypto/monero/xmr-eur-converter>
- [28] Defcon Monero Ring Signatures Presentation by Justin Ehrenhofer. Disponible en:  
<https://www.slideshare.net/JustinEhrenhofer/defcon-monero-ring-signatures-presentation-by-justin-ehrenhofer-2018>. Último acceso: Julio 2020

## ANEXO A - TRANSACCIÓN EN MONERO

Consideremos una transacción desde Alice hacia Bob. Las cuatro claves de Bob se denominarán de la siguiente manera:

$k_{B1}$  : Clave privada de visualización

$k_{B2}$  : Clave privada de gasto

$K_{B1}$  : Clave pública de visualización

$K_{B2}$  : Clave pública de gasto

Alice genera un nonce  $1 < r < N$  donde  $N$  es el orden de la curva elíptica. A continuación genera la clave pública de output  $K_o$  utilizando las dos claves públicas de Bob.

$$K_o = Hn(rK_{B1})G + K_{B2}$$

Pero, normalmente, una transacción contiene más de un output, y es necesario que cada uno de ellos tenga asociado un par de claves diferentes. Para ello, se concatena el índice  $i$  ( $i = 1, i = 2$ , etc) del output a  $rK_{B1}$  con tal de obtener hashes diferentes:

$$K_{oi} = Hn(rK_{B1} + i)G + K_{B2}$$

Alice añade  $rG$  en la transacción ( $rG$  es la clave pública de transacción). Por su parte, Bob tiene su wallet escuchando todas las transacciones que se publican en la blockchain. Para cada transacción, calcula  $K_{oi}$  utilizando la clave pública de transacción, su clave pública de gasto y su clave privada de visualización:

$$K_{oi} = Hn(rGk_{B1} + i)G + K_{B2}$$

Si el valor de esta clave pública calculada coincide con la clave pública de output en la transacción, significa que Bob es el destinatario de ésta. Entonces, puede calcular la clave privada del output:

$$k_{oi} = Hn(rGk_{B1} + i) + k_{B2} = Hn(rK_{B1} + i) + k_{B2}$$

Nótese que únicamente Bob puede calcular esta clave privada, ya que para calcularla necesita su clave privada de gasto.

## ANEXO B - OPCIONES DEL DEMONIO

Los parámetros que pasamos al demonio para arrancar un nodo local son los siguientes:

**--testnet:** Especifica que el nodo se conectará a una red de pruebas. Como nosotros por el momento trabajamos con unos nodos que no disponen de la blockchain de la testnet oficial, crearán una desde el bloque origen y será solo nuestra.

**--p2p-bind-port:** El puerto en el que escuchará el demonio a los otros demonios para sincronizarse entre ellos.

**--rpc-bind-port:** El puerto en el que escuchará la API del demonio. En el siguiente apartado se habla con más detalle de esta API.

**--zmq-rpc-bind-port** El puerto en el que escuchará la API ZMQ-RPC. No la usaremos puesto que no ofrece ninguna ventaja respecto RPC, pero no se puede deshabilitar.

**--no-igd:** Deshabilita UPnP en el router.

**--hide-my-port:** Desactiva la anunciación como candidato P2P hacia otros nodos.

**--data-dir:** El directorio donde se almacena la blockchain.

**--p2p-bind-ip:** La IP en la que se escuchará. Por defecto escucha en localhost, pero necesitamos que escuche en otras interfaces para el correcto funcionamiento del escenario.

**--log-level:** El nivel de logging mostrado por terminal y guardado en los archivos de log. El mínimo es 0 (warnings) hasta un máximo de 4 (trazas).

**--add-exclusive-node:** Añade otros nodos con los que el nuestro se emparejará. Hemos añadido otros dos demonios que tendremos corriendo en los puertos 38080 y 48080 simplemente para verificar que la conexión con otros nodos se realiza de manera correcta, pero por el momento con un solo nodo tendremos suficiente para nuestro escenario.

**--fixed-difficulty:** La dificultad que supondrá minar un bloque según lo define el protocolo de consenso Proof of Work en Monero. Hemos establecido el valor en 3000 de manera empírica. Con la capacidad de cómputo de la máquina utilizada permite minar un bloque (cuando solo hay una wallet minando) cada aproximadamente 20 segundos, valor suficientemente alto para no provocar desincronización (cuando haya más de una wallet minando) pero

suficientemente bajo como para poder aumentar la altura de la blockchain en un tiempo aceptable.

**--log-file:** El archivo en el que se escribirán los logs. Los necesitaremos posteriormente para analizarlos.

**--confirm-external-bind:** Confirma que *--p2p-bind-ip* estará escuchando en una IP diferente a 127.0.0.1.

En la documentación oficial del daemon [16] se pueden consultar todos los parámetros disponibles.

## ANEXO C - SCRIPTS PARA TRABAJAR CON WALLETS

El script de la Fig. C.1 sirve para crear una wallet con el nombre indicado como argumento.

```
import os
import sys
import random
import binascii

try:
    user = sys.argv[1]
except IndexError:
    user = str(input('User: '))

WALLET_DIR = "/home/samuel/testnet/wallets/"

if os.path.exists(WALLET_DIR + user + '.bin.address.txt'):
    sys.exit(sys.exit('Wallet already exists'))

word_list = [ ... ]

seed = ""
checksum = ""

for i in range(0, 24):
    word = random.choice(word_list)
    seed = seed + word + " "
    checksum = checksum + word[0:3]

checksum_word = seed.split()[int(binascii.crc32(checksum.encode())) % 24]
seed = seed + checksum_word

print('Mnemonic seed:')
print(seed)

BINARIES_DIR =
"/home/samuel/Developer/monero/build/Linux/master/release/bin/"
CMMND = "{0}monero-wallet-cli --testnet --generate-new-wallet {2}/{1}.bin
--restore-deterministic-wallet " \
        "--electrum-seed=\"{3}\" --password \"\" --log-file {2}/{1}.log;"

os.system(CMMND.format(BINARIES_DIR, user, WALLET_DIR, seed))
```

**Fig. C.1** Script para crear una wallet

Y el script para abrir una wallet creada se muestra en la Fig. C.2.

```
import os
import sys

WALLET_DIR = "/home/samuel/testnet/wallets/"
try:
    user = sys.argv[1]
except IndexError:
    sys.exit('Please introduce wallet name')

user = user.lower()
if not os.path.exists(WALLET_DIR + user + '.bin.address.txt'):
    sys.exit(sys.exit('Wallet does not exist'))

BINARIES_DIR =
"/home/samuel/Developer/monero/build/Linux/master/release/bin/"
CMMND = "{0}monero-wallet-cli --testnet --daemon-address 127.0.0.1:28081 "
\
    "--trusted-daemon --wallet-file {2}/{1}.bin --password '' --log-file
{2}/{1}.log --log-level 4"

os.system(CMMND.format(BINARIES_DIR, user, WALLET_DIR))
```

**Fig. C.2** Script para abrir una wallet

De manera que ahora para abrir una wallet (por ejemplo la de Alice) simplemente habrá que hacer:

```
/home/samuel/testnet$ python3 open_wallet.py alice
```